APPENDIX B

This Appendix contains a marked up copy of the original specification depicting the amendments thereto:

STRATEGIC PLANNING AND OPTIMIZATION SYSTEM

RELATED INVENTION

[0001] This application is a continuation of and claims the benefit of U.S. Patent Application Serial No. 09/951,334, filed 10 September 2001, still pending, which is specifically incorporated herein, in its entirety, by reference, which in turn is a continuation-in-part of U.S. Patent Application Serial No. 09/084,156, filed May 21, 1998, now U.S. Patent No. 6,308,162, and which in turn claims the benefit of U.S. Provisional Application No. 60/049,948, filed May 21, 1997 and U.S. Provisional Application No. 60/049,826, filed May 21, 1997 all of which are incorporated by reference herein.

BACKGROUND OF THE INVENTION

[0002] Area of the Art

[0003] The present invention relates to planning models, and more particularly, to a method for optimizing a planning model while managing strategic objectives.

[0004] Description of the Prior Art

[0005] As information technology continues to penetrate into all aspects of the economy, a wealth of data describing each of the millions of transactions that occur every minute is being collected and stored in on-line transaction processing (OLTP) databases, data warehouses, and other data repositories. This information, combined with quantitative research into the behavior of the value chain, allows analysts to develop enterprise models, which can predict how important quantities such as cost, sales, and gross margin will change when certain decisions, corresponding to inputs of the model, are made.

These models go beyond simple rules-based approaches, such as those embodied in expert systems, and have the capability of generating a whole range of decisions that would not otherwise be obvious to a designer of rules-based systems.

[0006] There is however a problem with the use of model-based decision-making tools. As the decision-making process is automated, the operational decisions that are recommended by the model may begin to deviate from broader considerations that are not specifically built into the enterprise planning model. The reason for this is that an economic model can realistically succeed only on either a small scale or large scale, but not on both. Incorporating both small scale decisions and large scale decisions into a single enterprise planning model would result in a model of enormous complexity, making the optimization of the enterprise planning model computationally impractical, and economically inefficient.

[0007] The importance of this problem can be illustrated with an example from the retail industry. A retailer can use a demand model to accurately forecast each item's unit sales given the item's price and other factors. However, if the demand model is used directly to optimize pricing decisions, it will generate prices that vary greatly from those of a human pricing manager. This is because a demand model has no knowledge of the enterprise's strategic objectives, and therefore generates prices that do not reflect the company's overall strategic pricing policy. That is, a business enterprise does not blindly set prices with an aim towards a maximum number of sales. Rather, each business follows an overall strategy that it hopes will allow it to succeed economically in the long run.

[0008] One example of business strategy and strategic goals is the "price image" of a retailer. Some retailers position themselves as "high end" operations. They stock more expensive goods and often a more extensive range of goods. Often the retail surroundings are expensive in appearance and a large amount of individual attention is given to each customer. The customer naturally expects prices to be higher at a "high end" retailer. On the opposite end of the scale is the low cost or discount retailer. Here the prices are lower—often much lower—but the stock selection may be limited, the surroundings are ~~Spartan~~spartan and individual customer service may be non existent. Well known economic models tell us that the lower prices of the discount retailer will result in sales of more units. Yet this does not necessarily translate into increased profit.

[0009] In maintaining a "high end" image, the retailer eschews the volume sales that discount pricing may afford. The higher prices stemming from a high price image operation may yield considerable profit. A careful cultivation of a particular price image can readily result in economic success both for the high end and the low end retailer. In the sense of economic ecology, such pricing strategies place the retailers in different niches so that they do not directly compete with each other. Maximizing the number of niches increases the possible number of retailers and maximizes the amount of money they can extract from the economy. This is but one example of a strategic decision or goal. An ordinary enterprise model cannot combine optimization of enterprise decisions with the retailer's strategic goals; consequently, the model's utility is greatly impacted. This inability to align and optimize an enterprise's operational decisions with its strategic objectives is a huge problem, and results in billion-dollar pricing inefficiencies in the retailing industry alone.

SUMMARY OF THE INVENTION

[0010] The present invention provides a computer-implemented method and system for controlling the optimization of a planning model while simultaneously satisfying at least one strategic objective. Until now, there has been no way for a manager to easily visualize the tradeoffs involved in setting different strategic goals. A pricing manager, for example, may need to understand the tradeoffs involved in driving towards a lower price image to compete with a key competitor. Hence, a question that arises is how ~~might~~ a lower price image might affect profits.

[0011] This is not a simple question. A pricing manager may be responsible for pricing over one thousand products. If he were to consider ten different possible prices for each product and then wish~~ed~~ to consider every possible pricing scenario (each scenario being a different combination of items with each item having a different possible price), the pricing manager would be faced with 10^{1000} different pricing scenarios. Considering that there are only about 10^{40} atoms in the entire universe it is not difficult to imagine that even the fastest computers can't ~~exploring~~explore all possible permutations in any reasonable amount of time.

[0012] ~~If one were~~Another question that arises is what it might look like to calculate and plot profits and price image for every possible pricing scenario~~, what would it look like?~~. Each pricing scenario, i.e., a unique set of prices for each product, has a price image and a profit associated with it. Referring to the Fig. 1, each pricing scenario corresponds to one point in the profit vs. price image graph. There are three important regions labeled in the graph ~~below~~of Fig. 1: A) inefficient pricing, this indicates that a different mix of prices can achieve the same price image with greater profits; B) unachievable, this indicates a level of profits that is unachievable through pricing manipulation alone; and C) an envelope of optimum pricing scenarios. This envelope of optimum pricing scenarios is what the manager needs to visualize to make an informed pricing decision.

[0013] Fig. 1 also represents the already mentioned relation between profit and price image. In the graph, the vertical axis represents profit from each scenario (sale of a group of priced items) while the horizontal axis represents price image (that is, an aggregate higher price). As the aggregate price increases, the profit increases until the overall high price begins to impact sales. The optimum pricing envelope indicates the highest profit scenario for each different gradation of price image. This makes the profit tradeoffs very clear regardless of the price image strategy (strategic objective) that is chosen by a particular retailer.

[0014] There are a number of strategic objectives used in business planning. In retail industries, many of these objectives are related to price. Price index is a commonly used measure that can have strategic import. A price index is simply a direct mathematical comparison between a retailer's prices and that of a competitor. This does not involve any psychological effects of pricing on the shoppers. To some extent this is a strategic objective based on the old maxim "all the market will bear." The tradeoff is between a lower price index (lower prices) and a higher profit. If a retailer's price index is consistently below his competitors, his profits per unit sale will be lower although increase in sales volumes may increase overall profits. The Price Index takes the general mathematical form of:

$$\phi = \frac{1}{N}\sum_{i=1}^{N} \frac{P_i}{P_i} \times w_i$$

[0015] Here the sum is over all prices and the weight factor, w_i is usually either a dollar or a unit sales factor.

[0016] However, simple Price Indices have two major flaws: 1) they infer how shoppers might respond to relative price differences, but they don't measure actual responses; and 2) Price Indices are not defined for "blind items"—that is, items for which the retailer does not know the competitors' prices. Since such competitive data can be difficult and/or expensive to collect, it is not unusual that a retailer may have little more than a small sample of his competitors' data.

[0017] The solution to this problem is to express the Price Index in the form of the closely related Price Image. The overall concept of Price Index has already been explained. However, in the present invention Price Image can readily be expressed in the following form:

$$\text{Price Image} = \text{SUM}(w_i * (US_i\ [C_i] - US_i\ [P_i])$$

[0018] Here the sum is over all prices, w is a weight, US[P] is the forecasted unit sales at price P, and C is the cost. As the price of an item is decreased, the unit sales increase and the overall price image decreases. When all items are sold at cost ($P_i = C_i$), the price image is zero. Because the parameters of a demand model are tuned using historical sales data, such a model forecasts from previous responses how shoppers will respond to price changes and can be defined with or without data from competitive prices ("blind items").

[0019] Service Time is yet another strategic objective.. Service time is a measure of how long it takes a given customer to obtain service. The objective of decreasing service time will increase operational and capital expenses. For example, in a retail setting, such as a busy supermarket, decreased service time requires more checkout lines with a concomitant increase in operational costs (more checkers) and capital expenses (more point of sales terminals). Yet decreased service time may well result in increased sales volumes. These same factors apply to non-retail business such as service or repair industries.

[0020] Another strategic objective is that of Risk. This is a measure of potential costs to a company. Risk could come from purchasers returning products, defaulting on loans or collecting on guarantees. Here the strategic decision is between decreasing the risks and increasing the profits.

[0021] Product Availability is a strategic objective or decision that can affect sales and profits independent of pricing strategy. Product availability is a measure of how often a product is available for immediate purchase. Certainly, a consumer is not pleased to discover that the desired beach stool is not available. This unavailability may well lose the immediate sale of the unavailable stool. However, the customer may well leave the store without purchasing other items they would have purchased had the item been available. However, increasing product availability generally increases carrying and inventory costs. So the tradeoff is increasing availability versus minimizing carrying and inventory costs.

[0022] A related strategic objective is that of Product Selection. This is a measure of the variety of products or services that a company offers. Increased product selection may result in increased purchases by customers on one hand and higher inventory, space and operations costs on the other hand.

[0023] Market Share is a strategic objective that measures the fraction of a market that is purchased from a given company as opposed to aggregate purchases for all competitors. Here the strategic decision is increasing market share versus increasing profits.

[0024] Revenue is a strategic objective that is a measure of all incoming money. Companies generally wish to increase overall revenue. This is related to market share, and the tradeoff is that of increasing profits versus increasing overall revenue. Fig. 2 shows the steps in an automated optimization of revenue. The graph represents the relationship between product margin (profit per sale) and revenue. As the margin decreases (*e.g.*, the price decreases), the overall revenue initially increases as sales volumes increase. Eventually revenue remains stationary as the margin continues to decrease because increases in sales volume do not compensate for the loss in per unit revenue (*e.g.* lower price per unit). The optimization routine successively analyzes the margin/revenue envelope to determine the scenario that gives the highest margin (profit) and the highest revenue.

[0025] The basic invention comprises a sophisticated enterprise planning model that allows a user to optimize and strategically manage an enterprise based on a variety of decisions. The user will select a planning model which will include many different types of decisions. Pricing is not necessarily a component of a planning model. A logistics planning model may only consider which route to take—that is many non-price related decisions can and do have major impacts on revenue and profitability.

[0026] In configuring the invention the user must first select and define a primary objective or goal. The most common primary objective is profit, although overall revenue, market share, risk-adjusted income as well as related factors that are user defined can form the primary objective. In the present invention it is possible to set up multiple primary objectives which can be weighted or treated in order. Within these primary objectives, negative or constraint factors can be treated as well as positive factors such as profit. The objectives contain or subsume decision variables that must be optimized to attain the objective. For example, a major decision that affects profit is price. A planner sets the price of each item based on actual constraints (like the actual cost of the goods) with a view towards maximizing profit. A planning model can simulate a large number of possible price decisions and based on real historical sales data can predict the decisions (prices) to reach the primary objective (produce the optimum profit).

[0027] After the primary objective(s) is selected, the present invention allows the optimization of the selected decisions in view of one or more strategic objectives. The general subject of strategic objectives has been discussed above. Unlike the primary objective and the tactical or physical constraints, strategic objectives are not fixed or known in advance. Strategic objectives depend on the manager's judgement; they represent strategic decisions where the manager must consider one or more tradeoffs that the manager may ~~chose~~choose to take, with the idea that the tradeoff will actually produce a benefit in the long run. As explained in regard~~s~~ to Fig. 1, the invention calculates a large number of scenarios and presents the results in a graphic form so that the optimum decision envelope can be visualized for the selected primary objective(s) in light of the selected strategic objective(s). This allows the manager to see the tradeoff involved with the selected strategic objective(s). A number of possible strategic objectives (including ~~weighed~~weighted combinations of strategic objectives) can be compared so that the manager has advance~~d~~ knowledge of the cost of the strategic decisions made. The model helps to select the strategy based on the cost of the strategy, but the actual strategic planning and its implications depend on the skill and the knowledge of the manager.

[0028] It should be apparent that one of the strengths of the current invention is the flexible tool it provides to the managers. An additional flexibility is provided in the choice of the ~~Optimization Method~~optimization method used to produce the various price scenarios of the ~~Enterprise Planning method.~~ enterprise planning model. There are a large number of optimization methods known in mathematics. The best optimization method depends upon the characteristics of the specific enterprise model being implemented. For this reason it can be advantageous to allow the user to ~~chose~~choose the optimization method and to even compare the results of various ~~different~~ optimization methods.

[0029] In general, enterprise planning models can be ~~quiet~~quite complex where the decision variables are non-linear, coupled, and discontinuous if not discrete. For this general class of problems common optimization algorithms are Ant Algorithms, Genetic Algorithms, Simulated Annealing, and others such as Evolutionary Methods, Branch and Bounds, Clustering Methods and Tabu Search Methods. New algorithms are constantly being developed and the present invention is designed ~~to~~ so that the user can incorporate user defined optimization algorithms.

[0030] For specific enterprise planning models, there can be much more efficient ways of optimizing than these general algorithms. Take, for example, an enterprise planning model where both the primary objective and strategic objectives have no terms that couple the decision variables. Since there is no coupling, each decision variable can be optimized independently, saving a tremendous amount of time. In some cases it is even possible to produce simple analytic expressions for the optimum values.

[0031] The present invention is preferably used through a graphic users interface. In a preferred embodiment, a user is presented with a menu on a display device. Using an input device, the user first selects a primary goal(s)/objective(s) to be realized—*e.g.*, maximize gross profits. The primary goal or objective is represented by a primary objective function which is dependent upon a set of operational variables. Each of the operational variables represents a single operational decision that the user seeks to optimize in order to reach the primary goal. Next, the user optionally selects one or more strategic objectives that the user would also like to be realized. The strategic objectives can be represented by constraint functions that are dependent upon a subset of the same set of operational variables as used by the primary goal.

[0032] Next, an effective objective function is constructed by combining the primary objective function with the strategic objectives, each being multiplied by a weighting factor. The resulting effective objective function depends on the same set of operational variables. The effective objective function is then optimized with respect to each of the decision variables, with the enterprise data providing physical constraints on the optimization process. The precise optimization process depends on the user's selection of optimization methods. As a result of the optimization, optimal values for each of the decision variables is obtained. The optimal values of the decision variables represent a set of operational decisions that should achieve the primary objective and the strategic objective.

[0033] The effective objective function can be optimized through a range of values of the weighting factor, with the results stored in a table. This computed table essentially provides a relationship between different optimized values of the primary objective, the strategic objective, and the values for the decision variables. These data are conveniently represented in a graphic form as in Fig. 1. The optimal pricing envelope clearly shows the user the effect of the strategic objective. The user is thus provided with a way to specify a target value for the strategic objective to attain and can use the table to interpolate the value for the weighting factor that corresponds to the target value. This interpolated value for the weighting factor is then inserted into the effective objective function. The effective objective function is optimized, yielding the set of operational decisions which optimize the primary objective function while at the same time satisfying the constraint(s) of the strategic objective(s) allowing the manager to readily determine the costs and benefits of various possible strategic ~~examples~~objectives.

BRIEF DESCRIPTION OF THE DRAWINGS

[0034] Figure 1 shows a graphic representation of an optimum pricing envelope displaying profit versus a strategic objective (price image);

[0035] Figure 2 shows a display of an optimum pricing envelope with a different strategic objective (revenue);

[0036] Figure 3 is a high level block diagram of a general purpose computer system used with the present invention;

[0037] Figure 4 is a picture of an example of an input menu displayed on a display device;

[0038] Figure 5 is a flowchart describing the overall operation of the system;

[0039] Figure 6 is a flowchart of a preferred embodiment of the Function Selection routine;

[0040] Figure 7 is a picture of an example of the input prompts displayed on the display device in a preferred embodiment of -the Function Selection routine;

[0041] Figure 8 is a flowchart of a preferred embodiment of the Constraint Mapping routine;

[0042] Figure 9 shows examples of data structures stored in memory for a Constraint Overview table and a corresponding list of bounds;

[0043] Figure 10 is a picture of an example of the input prompts displayed on the display device in a preferred embodiment of the Constraint Mapping routine;

[0044] Figure 11 is a flowchart of a preferred embodiment of the Preprocessing routine;

[0045] Figure 12 is a flowchart of a preferred embodiment of the Targeting routine;

[0046] Figure 13 is a flowchart of a preferred embodiment of the Bounding routine;

[0047] Figure 14 is a flowchart of a preferred embodiment of the Interpolation routine;

[0048] Figure 15 gives a schematic of the determination of quantities used for the interpolation of the Constraint Overview table; and

[0049] Figure 16 gives a graph presented on the display device of the data contained in an example Constraint Overview table, and an example of the targeting of a particular Price Image.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0050] The following description is provided to enable any person skilled in the art to make and use the invention and sets forth the best modes contemplated by the inventor of carrying out his invention. Various modifications, however, will remain readily apparent to those skilled in the art, since the general principles of the present invention have been defined herein specifically to provide an automated Strategic Planning and Optimization System.

Planning Model

[0051] The present invention provides an automated system for implementing a strategic planning model. Such models allow optimizations based on a variety of decision variables. Essentially, these variables can be any of a large number of decisions that must be made in planning a business. The following discussion should not be considered as exhaustive, but price is certainly one of the most apparent decisions that most business planners face. Other decision variables include promotional media (how should the product or products be promoted–how much should be spent–what approach should be used, etc.). In terms of a profit objective if too much money is spent on promotion, profits may be reduced, but if too little is spent sales will doubtless be decreased. Similar decisions revolve about promotion date (how soon before a sales event should the promotion be launched) and promotional duration (how long should the promotion continue) as well as promotional discount (which items should be put on sale and how much should the price be decreased below the usual optimized price).

[0052] There are a variety of other decision variables that can be made and modeled. Especially where particular goods are seasonal or likely to have model changes or otherwise become obsolete, the quantity of the good that should be purchased becomes a decision variable. Too large of a purchase will result in remainders that must be sold at a discount or a loss. Too small of a purchase may make it impossible to satisfy demand with the result that customers take their business elsewhere. For seasonal goods the decision variable is the question of when the goods should be purchased by the retailer. Peak season prices may be higher. Very early purchases may have lower purchase prices but capital is expended prematurely and the goods may then have to be stored before actual sale. Late purchases may have very attractive prices but may miss the peak demand by the customers. Similarly, it is important to determine the product location within the store and the amount of shelf allocated to a particular product and/or to a particular class of products. Overall selection of products is an important decision variable. How many different products within one product class should be stocked. How many different product classes should the establishment cover. There are many additional decision variables that may be important to particular users. The present invention permits ready customization so that other user specific decision variables can be added.

Objective and Constraints-Aggregate Measures

[0053] A basic planning model uses historical data to simulate the outcome of altering one or more decision variables. Generally, the interactions of the decision variables are expressed in terms of a primary objective (such as profit) that itself is often an aggregate measure of a number of separate decision variables. That is, the ultimate "product" of the model is suggested values for one or more decision variables; however, the goal or primary objective that invokes the decision variables is a global aggregate measure that subsumes the individual decision variables.

[0054] For convenience of the user, the system lists the various aggregate measures in an Aggregate Measures table from which the user can make a selection for any particular simulation. Examples of such aggregate measures include:

[0055] Profit

[0056] Cost

[0057] Revenue

[0058] Price Index

[0059] Price Image

[0060] Service Level

[0061] Marginal Cost of Revenue

[0062] Marginal Cost of Image

[0063] Risk-adjusted Income

[0064] Number of Price Changes

[0065] Shelf Length

[0066] Number of Displays

[0067] User Defined

[0068] The User Defined Measures include a weighted mix of any of the previously defined aggregate measures.

[0069] Note that the task of selecting the Primary Objective from the Aggregate Measure Table may also include the further task of selecting whether the Objective is to be maximized or minimized. Strategic Objectives are also included in the Aggregate Measure Table and are selected by the user. The system gives the user the option of ranking the multiple Strategic Objectives in terms of weights to prioritize multiple strategic objectives or in terms of a target value for a particular Strategic Objective. When presented with a target value for a Strategic Objective, the system operates to find the proper weight for the Objective that will yield the target value after optimization. When presented with the weight of a Strategic Objective, the system proceeds to optimize the model in light of that weight.

[0070] Finally, some of the entries on the Aggregate Measures Table are neither decision variables *per se* nor Strategic Objectives *per se*. Rather they are Tactical Constraints which operate as decision-level constraints with a possible strategic import. Typical Tactical Constraints include a maximum or minimum price for an item or class of items, and a defined relationship between prices (for example, the price of item *n* must be less than or equal to the price of item *m*. Overall ceilings or floors can also be set for Tactical Constraints; for example, the system can be constrained so that the overall price change is less than a given percentage.

<u>Optimization Methods</u>

[0071] The automated planning model operates by calculating the outcomes for a large number of individual scenarios involving the selected decision variables as determined by items from the Aggregate Measures Table as well as Strategic Objectives and Tactical Constraints. Various Optimization Methods or algorithms can be employed. The best choice of optimization method depends on the characteristics of the specific model being implemented. In most cases the best result is obtained by allowing the user to select several optimization methods and to compare the results obtained by using a variety of methods on the same data set (goals and strategic objectives from the Aggregate Measures Table in combination with actual historical sales data). The general optimization methods include Ant Algorithms, Genetic Algorithms, Tabu Searches, Branch and Bound, Clustering Methods and Simulated Annealing. Because new optimization schemes are constantly developed, the system allows for the incorporation of any user defined Optimization Method.

[0072] For specific planning models, methods that are more efficient than the general optimization methods may exist. In an enterprise planning model where both the primary objective and the strategic objectives have no terms that impact the same decision variables, each decision variable can be independently optimized, thereby saving a tremendous amount of time. In some cases it is even possible to produce a simple analytic expression for the optimum values.

[0073] As shown in Figure 3, a system includes: an input device 101 such as a keyboard, through which a user enters commands, inputs functions, etc.; a display device 102 for displaying tables, etc.; a storage device 103 such as hard disk drive for storing results and enterprise data; a memory 104 for storing program instructions, tables and results; and a processor 105 for performing various kinds of processing and controlling the overall operation of the system. These instructions include, for example, a Constraint Mapping routine 112, a Preprocessing routine 113, and a Scenario Analysis routine 114. Tables and results may include, for example, data lists 115 and a table portion 116. Table portion 116 may have a Constraint Overview table, and Optimum Value table, and/or a Target Value table stored therein in response to the execution of the aforementioned instructions. Of course, this representation is merely schematic and all or part of the system may exist as a network device.

[0074] It will be understood that the described embodiments of the present invention are embodied as computer instructions stored in memory 104 and executed by processor 105. These instructions can also be stored on a computer readable medium, such as a floppy disk, CD ROM, etc., and can also be transmitted in a network such as the internet, an intranet, etc., via a carrier wave embodying the instructions.

Operation of the Invention

[0075] The operation of a preferred embodiment of the present invention will now be described in brief with reference to Figs. 4 and 5 before being described in detail with reference to Figs. 6 to 9.

[0076] A menu as shown in Fig. 4 is presented to the user on the display 102. At this time, the user enters one of the following selections through the input device 101: '1' to select from the Aggregated Measure Table, '2' to perform ~~the~~

constraint mapping routine 112, '3' to perform ~~the~~ preprocessing routine 114, '4' to perform ~~the~~ scenario analysis routine 114, '5' to output results to the storage device 103, and 'Q' to terminate use of the system. Other appropriate user interfaces covering other methods and formats of input can, of course, be used.

[0077] The processor 105 receives the entered information, and the situation of the system is passed to one of the appropriate steps described below, according to the inputted value. This is represented schematically in Figure 5.

[0078] (Step 1001): Goal/Objective Selection

[0079] At this step, the user selects a primary goal or objective to be analyzed, along with secondary goal(s) and strategic objective(s). A convenient manner of making the selection is use of the Aggregate Measures Table. The system is aware of the composite nature of many of the entries and derives multiple goals and strategic objectives from one or more table entries. Alternatively, the goal(s) and strategic objective(s) can be individually selected from separate Goals and Strategic Objective tables. The details thereof are discussed below in conjunction with Fig. 6. Normally the principal goal is denoted as the "Primary Goal or Objective". There may actually be more than one primary goal selected in which case the goals are either treated in order with successive optimizations or are given different weights and optimized simultaneously. Besides the simple or compound primary goal, there may be an additional Strategic Objective. With a Strategic Objective, the Primary Goal is optimized for a Strategic Objective target or over a range of target values for the Strategic Objective. Keep in mind that the primary goal/objective is subject to actual physical limitations/constraints whereas there are no direct physical constraints on the Strategic Objective. By looking at how the Primary Objective is altered by the Strategic Objective, the manager obtains a clear picture of the economic cost of implementing the Strategic Objective. The Strategic Objective may also be compound wherein the various strategic components are given different weights.

[0080] (Step 1101)

[0081] After the user selects the primary goal to be realized—*e.g.*, maximization of gross profits~~. This~~—, the primary goal is represented by a primary objective function ~~Π which~~, Π. The primary objective function, Π, depends upon a set of variables $\{X_i\}$, each of which represent a single operational decision. For example, in the field of retail, a primary goal is normally the gross profit,

$$\Pi = \sum_i Q_i \left(P_i - C_i\right),$$

[0082] where $Q_i = Q_i\ (P_i)$ is the predicted demand Q_i for an item *i* based on its price P_i, and C_i is the item's cost. In this case, the variables $\{X_i\}$ would be the set of all prices $\{P_i\}$. The primary goal may be defined by any model that attempts to optimize many operational decisions, *i.e.*, those decisions that occur on a lower level. In a preferred embodiment, a plurality of objective functions corresponding to each of a plurality of predetermined goals will be stored in storage 103, and provided to the user on display device 102. However, it is anticipated that the user can modify existing goals and/or create new goals.

[0083] (Step ~~1102~~1002)

[0084] Where the user has selected a Strategic Objective, this acts as an additional constraint on the enterprise model. This constraint should represent some global, large-scale objective that is not included in the primary objective function Π that provides the definition of the primary goal. The Strategic Objective is represented by a constraint function ϕ, and should depend on the same set of variables $\{X_i\}$ that the primary objective function Π depends upon, or some subset thereof. Ideally the constraint function ϕ should be defined so that it reflects some aggregate property that the variables should attain. Significantly, the constraint function ϕ can be virtually any function that the user feels is important.

[0085] For example, the equation for the gross profit, which is given above, can be used as the primary objective function whose value is maximized by adjusting prices on all items. Once maximized, the result is a set of prices for each item that maximizes the overall gross profit. On the other hand, the user might also like to set prices so as to achieve a particular level of sales—i.e., a Strategic Objective of achieving a particular level of sales. A suitable strategic constraint function for the total amount of sales, can be defined as

$$\phi = \sum_i Q_i P_i$$

[0086] where Q_i and P_i are defined as above. This constraint function depends on all prices and demands, and for a given value of total sale, there could be many combinations of quantities (Q_i) and prices (P_i) that would give the same answer. However, the actual combination chosen to optimize total sales will depend upon the optimization of the primary objective function Π, as will be discussed below.

[0087] The primary goal/objective of the present invention can be any standard goal (or a compound goal) of an enterprise planning model, such as the maximization of gross profits. The Strategic Objective (or compound strategic objectives) can be any strategic factors that the user seeks to analyze in conjunction with the primary goal/objective, for example, increase or decrease of price image. As an example, a retail pricing manager may seek to set prices such that gross profits are maximized while at the same time, meeting the store's other long term goals, such as maintaining a particular price image.

[0088] It will be appreciated by those having ordinary skill in the art that prior art enterprise planning models are limited by the physical constraints of the enterprise planning model. Thus, the operational decisions that are recommended by the model will likely deviate from strategic considerations that are not specifically built into the enterprise planning model. This is a primary reason that retailers have traditionally avoided the use of demand models to help them price their products; namely, the results cannot reflect the company's overall strategic policies. As will be appreciated by this discussion, by incorporating strategic objectives into the enterprise model, the present invention provides an enterprise planning model that goes far beyond the physical constraints of traditional enterprise planning models.

[0089] (Step 1002): Constraint Mapping

[0090] With a Primary Goal and a Strategic Objective the behavior of the Primary Goal is determined over a range of values of the Strategic Objective. In the situation mentioned above above, gross profits would be maximized (optimized) over a range of expected market share values. These data points are then recorded in a Constraint Table that is used to set bounds for the model. The details of ~~the~~ Constraint Mapping routine 112 (Fig. 3) are discussed below in conjunction with Fig. ~~13~~8.

[0091] (Step 1003): Preprocessing

[0092] To provide the system with an efficient method to analyze various scenarios for achieving the primary and auxiliary goals (*i.e.*, Scenario Analysis, discussed below), the data generated in the Constraint Mapping step are preprocessed. The details ~~thereof~~of Preprocessing routine 113 (Fig. 3) are discussed below in conjunction with Figure 11.

[0093] (Step 1004): Scenario Analysis

[0094] Next the system defines a set of scenarios, i.e., projected values for the Strategic Objective that the user would like to achieve. For each scenario defined, a set of operational decisions are provided that maximize the Primary Goal while simultaneously satisfying the Strategic Objective. This step is performed for each scenario selected by the user.

The details thereof are discussed below in conjunction with Fig. 12. Referring to the example provided above, the present invention provides the pricing manager with the necessary information to achieve both the Primary Goal (e.g., maximize gross profits) and a Strategic Objective (e.g., increase market share)—results that are not provided in prior art enterprise models.

[0095] (Step 1005): Output Results

[0096] The operational decisions, primary goal, and auxiliary goal determined for each scenario are placed in the storage device 103. Thus, in the retail pricing example given above, the retail pricing manager would be provided with the optimum prices for each item to be sold that would allow the store to meet both the Primary Goal of maximizing gross profits, and the Strategic Objective of increasing market share. By optimizing gross profit over a range of possible market shares the manager can see how much profit must be given up for each incremental increase in market share. This is vital in making informed strategic decisions.

Goal Selection

[0097] A preferred embodiment of this routine ~~will be~~ is described with reference to Figs. 6 and 7. The user is presented with a menu on display 102, such as illustrated in Fig. 7, to prompt the user through the Goal Selection routine as illustrated in Fig. 6. It should be appreciated that other appropriate methods and formats of input can, of course, be used, and that the menu presented in Fig. 7 is presented for illustrative purposes only.

[0098] The addition of a Strategic Objective to the enterprise model allows the user to analyze enterprise planning decisions otherwise not available in the prior art. For example, when pricing their products, retail pricing managers generally seek to have their prices reflect a certain image of their stores. A discount retailer would like its prices to be perceived as being lower than other retailers. This so-called "price image" is an example of a strategic constraint; it does not correspond to any physical constraint on the prices, and it does not directly correspond to any single decision made by a enterprise planning model. Instead, it is a function of all the prices in the market, and it represents a higher-level property that the pricing manager would like to be able to choose and control with precision.

[0099] As illustrated in Fig. 7, a preferred embodiment includes, in addition to other Strategic Objectives, a mathematical definition of the price image. Thus, the present invention could be used to control the prices predicted by any demand model to ensure that a particular desired price image is attained. A preferred definition of a price image is

$$\phi = \frac{1}{N}\sum_{i=1}^{N} \frac{P_i}{\overline{P_i}} \times w_i ,$$

[0100] where $\overline{P_i}$ is the average price of item *i* in the market of interest, w_i is a weighting function for item *i* and N is the total number of items in the model. The weighting function is suitably defined such that other factors can modify the contribution of a single item to the overall price image. For example, w_i could be proportional to the sales of the item, so that items that are not frequently sold do not influence the price image as much as items with high sales. In the absence of any relevant information, the weighting functions may simply be set to 1. It should be apparent that other definitions, such as the one presented earlier, of price image can be utilized, and that the above definition is presented for illustrative purposes only.

[0101] The price image can be used in conjunction with the present invention to address a long-standing problem with retail demand modeling. Retailers have found that if a demand model is used to optimize prices on items to yield the greatest gross profit, the model will invariably choose prices that are higher than what a human price manager would have intuitively chosen. The typical outcome is that, in the short term, shoppers continue to buy products at these higher prices, and this does in fact yield a higher gross profit. However, over the long term, customers become aware that the price image of the store has risen, and eventually turn to other stores. Thus, controlling the price image from the outset can prevent this problem with different consumer responses on different time scales. By determining one's price image from existing prices, a retailer could then use a demand model, in the context of the present invention, to obtain greater profit even while maintaining the same overall price image.

Constraint Mapping

[0102] Because a strategic constraint (Strategic Objective), such as price image, does not represent a physical restriction on the system, it is not necessary that it be met rigorously. Rather, it is more desirable to vary the constraint over a range of scenarios, and then determine which set of predicted decisions aligns most favorably with the Primary Goal and the Strategic Objective. The object is to have control over the decisions being made, without being locked to a single set of decisions. For this reason, it is not practical to use conventional constraint-based optimizations, which are usually employed for physical constraints. A more efficient method for treating strategic constraints is described below.

[0103] By obtaining solutions over a broad range of scenarios, the user of the invention obtains a picture of how the optimal predictions vary according to changes in the desired large-scale goal. After seeing this picture, the user may target a specific large-scale scenario to be realized and subsequently obtain the set of decisions that are the most optimal, given the constraints of that particular scenario. The method can be used with a wide variety of models and objective functions.

[0104] The input to ~~this~~Constrint Mapping routine 112 includes the primary goal as represented by the primary objective function Π, the set of independent variables $\{X_i\}$ that affect the Primary Objective function Π, and a mathematical definition, *i.e.*, the constraint function ϕ for the Strategic Objective, all of which are stored in memory 104 and/or storage 103. A preferred embodiment of ~~this~~Constrint Mapping routine ~~will be~~112 is described with reference to Fig. 8.

[0105] (Step 1201)

[0106] At this step, the user is prompted to select the extent to which the Strategic Objective will affect the Primary

Goal. To achieve this, the user enters a minimum value ψ^{min}, a maximum value ψ^{max}, and the resolution $\delta\psi$ which represents step increments to be tested between ψ^{min} and ψ^{max}. Fig. 10 illustrates user prompts that may be displayed on display 102 in this step. The actual value of each of these variables will depend upon the particular situation being studied. To begin the constraint mapping, the value for ψ is initialized to ψ^{min}. In addition, the system can be allowed to automatically select a range of ψ^{min} to ψ^{max} that is based on general ranges shown to be useful.

[0107] (Step 1202)

[0108] A loop is begun in which the variable ψ takes on values between ψ^{min} and ψ^{max} incremented by $\delta\psi$.

[0109] (Step 1203)

[0110] The routine constructs an effective objective function:

$$\Pi_{eff} = \Pi - \phi\psi$$

[0111] It is important to note Π_{eff} depends on the same variables $\{X_i\}$ as the primary objective function, and represents an effective goal. As can be seen above, the effective objective function is constructed by taking the primary objective function and subtracting the constraint function as weighted by the value of ψ.

[0112] (Step 1204)

[0113] At this step, the effective objective function Π_{eff} is maximized with respect to all the independent variables, and the enterprise data is stored in the storage device 103. A useful method of maximizing Π_{eff} is the method of simulated annealing because it is one of the few techniques available for solving discrete, nonlinear, high-dimensional functions. This technique is known in the art and is documented in the following reference, which is herein incorporated by reference: W. *Press et. al.*, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, New York (1992).

[0114] The simulated annealing technique is well suited for this problem for several reasons. In a typical situation there are possibly thousands of independent variables that correspond to thousands of operational decisions, and there are very few techniques that can optimize an objective function with this many variables in an efficient amount of time. In addition, the primary objective function and the constraint (strategic) function will typically depend upon many discrete variables, for example, price which can only change in units of cents. The simulated annealing technique is able to handle this complication, and in fact is ideally suited for optimizations involving discrete variables. Other optimization routines can be utilized and may be more efficient in some situations, for instance, when the types of decisions that influence the objective function are captured by continuous variables, or when the system to be studied is very small.

[0115] The variable ψ serves the purpose of being a reward or a penalty. When the value of ψ is equal to zero, the effect of the auxiliary (strategic) function on the aforementioned optimization procedure is not felt at all, and optimization of the effective objective function amounts to an unconstrained optimization of the primary goal. Therefore, it would generally be useful to define ψ_{min}, ψ_{max}, and $\delta\psi$ such that ψ is zero during at least one point in the iteration procedure. If the value of ψ is large and positive, then the constraint acts as a penalty, and the optimization will be skewed towards a solution that results in a lower numerical value of the constraint function. If the value of ψ is large and negative, then the constraint acts as a reward, and the optimization will be skewed towards a solution that results in a higher numerical value of the constraint function. The magnitude of ψ serves to fix the relative weight of the constraint, and accordingly different values for ψ will result in different numerical values of the constraint that will be attained ~~which~~when the objective function is optimized.

[0116] (Step 1205)

[0117] The output from step 1204 is the maximized value of Π_{eff} and the resulting values for the independent variables $\{X_i\}$. These independent variables are stored in an Optimum Value table in ~~the~~ Table portion 116 (Fig. 3) of memory 104. The values of the constraint function ϕ and the primary objective function Π are determined from these variables, and subsequently, Π, ϕ, and ψ are all stored in the Constraint Overview table in the Table portion 116 of memory 104, as shown in Fig. ~~5B~~9.

[0118] Next, the value of ψ is incremented by $\delta\psi$, and a judgment is made as to whether ψ is greater than ψ^{max}. If it is not, the routine goes back to step 1202. If it is, the Constraint Mapping procedure terminates.

[0119] The information stored in the Constraint Overview table provides a concise summary of the behavior of the target market—i.e., a summary of the effect that the Strategic Objective will have on the Primary Goal. These data may be stored in a file or printed, or passed on to another routine. For example, since the table contains various Primary Goal values for each set of values determined from the Strategic Objective, data from the table may be used as input to a visualization routine or package. In one embodiment, the user is provided with an intuitive, graphical view of the dependence of the Primary Goal on the target value of the Strategic Objective, as illustrated in Fig. 16 or Fig. 1.

[0120] After obtaining data in the Constraint Overview table, and possibly visualizing it or comprehending it in some other manner, the user may choose to terminate the operation of the system, or proceed to ~~the~~ Preprocessing ~~Routine~~routine 113 (Fig. 3).

Preprocessing Routine

[0121] In one embodiment, before the data contained in the Constraint Overview table are used to generate a forecast for a specific scenario (see discussion of Scenario Analysis below), it is preprocessed into a computationally efficient form. This step generates information for use in subsequent operations. Without preprocessing, the subsequent Scenario Analysis routine would have to be performed in a much less efficient manner, and the additional computation time would likely be undesirable for the user. A preferred embodiment of

~~this~~Preprocessing routine ~~will be~~113 is described with reference to Figure 11.

[0122] (Step 1301)

[0123] A list $\{\psi_i^{extr}\}$ is created, and ψ^{min} is made the first entry in the list.

[0124] (Step 1302)

[0125] The values of ψ in the Constraint Overview table are scanned from ψ^{min} to ψ^{max}. Anytime an extremum is found, that is, a point where the constraint function ϕ attains a local minimum or maximum, the value of ψ at this point is added to the list $\{\psi_i^{extr}\}$. As will be discussed further below, the local minimums and maximums are obtained so that any value in the weighting range, ψ^{min} to ψ^{max} , can be efficiently interpolated.

[0126] (Step 1303)

[0127] ψ^{max} is made the last entry in $\{\psi_i^{extr}\}$.

[0128] The list $\{\psi_i^{extr}\}$ contains the ψ value of endpoints of successive segments in the Constraint Overview table where the constraint function ϕ representing the Strategic Objective, is monotonic increasing or monotonic decreasing. In the trivial case where the constraint function ϕ is monotonic increasing or monotonic decreasing throughout the entire list, then $\{\psi_i^{extr}\}$ contains only the lowest and highest values of ψ, respectively, in the Constraint Overview table.

Scenario Analysis Routine

[0129] An embodiment of ~~the~~ Scenario Analysis routine ~~will be~~114 (Fig. 3) is described with respect to Figure 12.

[0130] (Step 1401)

[0131] The user selects a set of scenarios—i.e., specifies values for the Strategic Objective that the user would like to see attained—for example, a particular gross margin or level of total sales.

[0132] (Step 1402)

[0133] Control of the system is first passed to the Bounding routine. The input to this routine is the Constraint Overview table obtained in ~~the~~Constraint mapping routine 112 (Fig. 8), and all the values of the constraint to be targeted, as well as the list $\{\psi_i^{extr}\}$. The outputs from the Bounding routine are the values ψ^{low} and ψ^{high} which correspond to the entries in the table which bound all target values of the constraint function ϕ.

[0134] ~~(Step 1403)~~If the values of ψ^{low} and ψ^{high} for a particular constraint target are null, this indicates that the desired scenario is not contained within the bounds of the Constraint Overview table. In this case, the situation of the routine skips directly to step 1406. If the user wishes to analyze the particular scenario that was rejected, then the Constraint Mapping routine may be run again, with different values of ψ^{min} and ψ^{max} in order to extend the range of the analysis. If this extended map still does not capture the desired scenario, then it is likely that the user has chosen to analyze a scenario that is impossible to attain.

[0135] ~~If a particular target value occurs in more than one place in the Constraint Overview table, that is, if there are multiple solutions, then each of these becomes a scenario of its own, and is added onto the list of targeted constraint values.~~ (Step 1403)

~~**[0136]** If the values of ψ^{low} and ψ^{high} for a particular constraint target are null, this indicates that the desired scenario is not contained within the bounds of the Constraint Overview table. In this case, the situation of the routine skips directly to step 1406. If the user wishes to analyze the particular scenario that was rejected, then the Constraint Mapping routine may be run again, with different values of ψ^{min} and ψ^{max} in order to extend the range of the analysis. If this extended map still does not capture the desired scenario, then it is likely that the user has chosen to analyze a scenario that is impossible to attain.~~

[0136] If a particular target value occurs in more than one place in the Constraint Overview table, that is, if there are multiple solutions, then each of these becomes a scenario of its own. These scenarios are added onto the list of targeted constraint values.

[0137] (Step 1404)

[0138] For each scenario that does not have null values for the bounds, the values ψ^{low}, ψ^{high}, the particular constraint target and the Constraint Overview table are passed to the Interpolation Routine. The output from this routine is an estimate of the value of ψ (denoted ψ^{est}) that, when used to optimize the effective objective function, will yield a value of the constraint function ϕ close to constraint target.

[0139] (Step 1405)

[0140] The effective objective function is constructed:

$$\Pi_{eff} = \Pi - \phi\psi^{est}$$

.

[0141] Π_{eff} is optimized with respect to all the independent variables. Again, an effective method is the simulated annealing technique, though others could be used. The output from the optimization routine includes the optimized values of the independent variables, such as the price and quantity for each item, and the resulting values of the objective function and constraint function.

[0142] The resulting constraint value is the one that most closely matches the target constraint value. The level of agreement will depend in part upon the nature of the system being analyzed and in part on the resolution of the mapping.

[0143] (Step 1406)

[0144] The values of the independent variables, the resulting objective function and the constraint function are stored in the Target Value table. A judgment is made as to whether all the scenarios have been analyzed. If they have not, the situation of the routine returns to 1404; otherwise, the Scenario Analysis routine 114 terminates.

[0145] Although in the routine described above, the Interpolation routine was used to obtain the estimate ψ^{est}, in an alternative embodiment this quantity may be determined by used of a root-finding technique. Many such techniques are well known in the art, and the particular choice will depend upon the known qualities of the system. One particular root-finding technique that is appropriate for discontinuous functions is the Van Wijngaarden-Dekker-Brent method, which is documented in W. Press *et. al., Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, New York (1992), and is herein incorporated by reference. The use of a root-finding technique is particularly desirable if the desired constraint target needs to be met with high accuracy. However, the root-finding technique will be computationally intensive for virtually every optimization of the effective objective function Π_{eff}. By contrast, the Interpolation routine makes use of data that have already been calculated and stored in the Constraint Overview table, making the scenario analysis computationally efficient.

[0146] Fig. 16 provides an example of how the predicted profits from a demand model could vary according to the price image of the particular group of items. By using competitive data, a retail pricing manager could find out the price image of all the other stores competing in the market with their store. For example, suppose the manager determines that the store should have a price image of -6.0 (measured relative to the market), this corresponds to choosing a value -6.0 from the horizontal axis, and then having the system optimize prices such that the point X on the graph is attained, realizing a profit of $38,000. Note that the manager can also use this display to determine how much more profit a less negative price image would yield.

Bounding Routine

[0147] The purpose of the Bounding ~~Routine~~routine is to determine the location in the Constraint Overview table in which the target value for the constraint functions can be found. The input to this routine is the Constraint Overview table, the constraint target value and the list of local minimums and maximums for ψ, denoted $\{\psi_i^{extr}\}$. The output from this routine is the entries in the table that bound these target values. A preferred embodiment of ~~this~~the Bounding routine ~~will be~~is described with reference to Fig. 13.

[0148] (Step 1501)

[0149] For each segment defined by $\{\psi_i^{extr}\}$, a bisection routine is performed to determine if the constraint target $\{\phi^{targ}\}$ is contained in that segment.

[0150] (Step 1502)

[0151] If the constraint target $\{\phi^{targ}\}$ is contained in that segment, then the bisection determines which entries in the Constraint Overview table corresponds to the bounds on the constraint target. These bounds, denoted as ψ^{low} and ψ^{high}, for the lower and upper bounds, respectively, are stored in a list $\{\phi_i^{bound}$~~}.~~} at a step 1503. If the constraint target $\{\phi^{targ}\}$ is not contained in that segment, then a null value is returned~~.~~ at a step 1504.

[0152] ~~A~~At a step 1505, a judgment is made as to whether all the listed segments defined by $\{\psi_i^{extr}\}$have been analyzed. If they have not, i.e., if there are no more segments, the control of the routine is returned to step 1501. If they have, $\{\phi_i^{bound}\}$ is returned to the calling routine, including the cases where these variables are null. Using the data from Fig. 9 as an example, $\{\phi_i^{bound}\}$ for a particular list of target values for ϕ is shown.

Interpolation Routine

[0153] This routine utilizes known interpolation techniques to interpolate a value of ψ from the Constraint Overview Table. The input to this routine includes~~of~~: the Constraint Overview table; the specified target values for the constraint functions, given by ϕ^{targ}, and the values ψ^{low} and ψ^{high} which bound the location in the table where the desired solution is to be targeted. The output from this routine is the value ψ^{est}, which is an interpolated value of a function $\psi(\phi)$ that is constructed from the part of the table containing ψ^{low} and ψ^{high}. In general, this interpolated value can be constructed from any prior art interpolation routine, as long as the routine makes use of the data in the Constraint Overview table that is near the entries ψ^{low} and ψ^{high}; otherwise, the accuracy of the interpolation will be compromised. Below is shown one embodiment of this interpolation routine with reference to Fig. 14.

[0154] (Step 1601)

[0155] The two values ψ^{low} and ψ^{high} are assigned to the variables α_2 and α_3, respectively. The values of corresponding entries of ψ in the table are assigned to β_2 and β_3, respectively. The value of the constraint function ϕ in the Constraint Overview table immediately below α_2 is assigned to α_1, and the matching value of the constraint function ϕ is assigned to β_1. The value of ψ in the Constraint Overview table immediately above α_3 is assigned to α_4, and the matching value of ϕ assigned to β_4. This is elucidated more clearly in Fig. 15. Note that this process is not affected by whether the Constraint Overview table is monotonic or not; the distinction is only used to determine whether or not there is a possibility of multiple solutions.

[0156] (Step 1602)

[0157] The values α_{1-4} and β_{1-4} are used to construct an interpolated function $\alpha(\beta)$. This fourth-order interpolation is then used to obtain an approximation $\psi^{est} = \alpha(\phi^{targ})$. An effective method for doing this is Neville's algorithm, which is described in W. *Press et. al., Numerical Recipes: The Art of Scientific Computing, Cambridge University Press*, New York (1992), and is herein incorporated by reference. The estimated value ψ^{est} is then returned to the calling routine.

[0158] If the values ψ^{low} and ψ^{high} are located near the ends of the Constraint Overview table, such that there does not exist two values in the table which are lower or higher than ϕ^{targ}, then the values for the α's and the β's would need to be chosen in a slightly different manner. If there is only one value of ϕ in the Constraint Overview table that was lower than ϕ^{targ}, then this one would be made β_1, the next three entries higher than ϕ^{targ} would be made β_2 through β_4, and the α's would be chosen accordingly. If there is only one value of ϕ in the Constraint Overview table that was higher than ϕ^{targ}, then this one would be made β_4, and the next three entries lower than ϕ^{targ} would be made β_1 through β_3, and the α's would be chosen accordingly.

[0159] If the Constraint Overview table contains fewer than 4 entries of ϕ, then the fourth-order interpolation would have to be replaced with a lower-order method, such as linear interpolation.

[0160] Having thus described a preferred embodiment of the Method for Controlled Optimization of Enterprise Planning Models, it should be apparent to those skilled in the art that certain advantages of the ~~within~~ method have been achieved. It should also be appreciated that numerous modifications, adaptations, and alternative embodiments thereof may be made within the scope and spirit of the present invention.

[0161] For example, the method described above may be extended to situations in which there is more than one Strategic Objective to be applied simultaneously. Instead of one constraint function ϕ, representing one auxiliary goal, there would be a set $\{\phi_i\}$ of them—one constraint function for each ~~Streategic~~Strategic Objective. Instead of a single variable ψ there would be a set $\{\psi_i\}$, each member of which corresponds to one of the Strategic Objectives. The effective objective function would thus be defined as:

$$\prod_{eff} = \prod - \sum_j \psi_j \phi_j,$$

[0162] and the map would exist in two or more dimensions, each of which corresponds to one of the Strategic Objectives. The values of the ψ_j would each be varied such that the multidimensional space spanned by them is captured by a discrete mapping within specified bounds ψ_j^{min} and ψ_j^{max} on each of the ψ_j. The simulated annealing technique could be used to perform the optimization of the effective objective function Π_{eff}. Finally, the Constraint Overview table would hold data for the entire multi-dimensional map.

[0163] For the Scenario Analysis routine, a scenario would include of a group of target values $\{\phi_i^{targ}\}_j$ that each of the constraint functions should attain simultaneously. The effective objective function would again be constructed in a manner similar to the one described above. The main difference for the multiple constraint implementation is the determination of $\{\psi_i^{est}\}_j$, which are the values for the $\{\psi_i\}$ that yield the targets $\{\phi_i^{targ}\}_j$. The Preprocessing, Bounding, and Interpolation routines would need to be adapted for multidimensional systems. However, once $\{\psi_i^{est}\}_j$ has been determined, the optimization of Π_{eff} is again performed to yield the values for the independent variables that yield the desired $\{\phi_i^{targ}\}_j$ while optimizing the primary objective function Π.

[0164] The following claims are thus to be understood to include what is specifically illustrated and described above, what is conceptually equivalent, what can be obviously substituted and also what essentially incorporates the essential idea of the invention. Those skilled in the art will appreciate that various adaptations and modifications of the just-described preferred embodiment can be configured without departing from the scope of the invention. The illustrated embodiment has been set forth only for the purposes of example and that should not be taken as limiting the invention. Therefore, it is to be understood that, within the scope of the appended claims, the invention may be practiced other than as specifically described herein.

APPENDIX C

This Appendix contains a marked up copy of the Abstract depicting the amendments thereto:

ABSTRACT OF THE DISCLOSURE

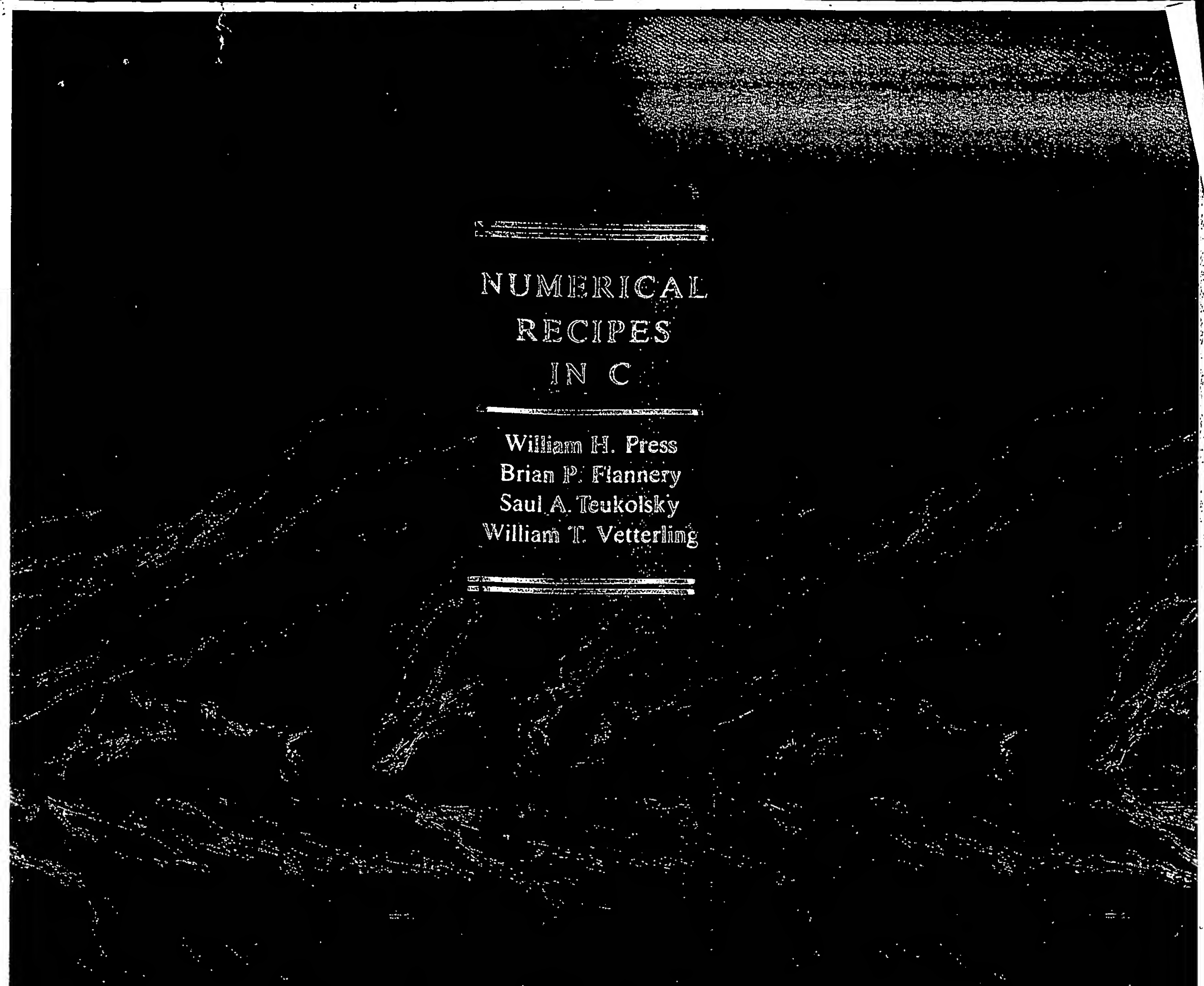
A software method for ~~controlling the optimization of a planning model that uses historical sales data to predicts optimal prices and similar factors for meeting a number of business goals. Unlike previous systems that allow a user to model prices and other factors based on physical constraints, the present invention allows the optimization to occur against the background of one or more strategic objectives. Such objectives, such a price image, are not set by physical constraints but instead are imposed by the user with the notion that they will provide a strategic and ultimately an economic advantage. The system allows the analysis of the costs and benefits of such management imposed strategic objectives~~ strategic planning and optimization allows a user to model an enterprise to visualize an effect of a strategic constraint on a primary goal of the enterprise. A primary goal of the enterprise is selected, and is represented by a primary objective function which, in turn, depends upon a set of operational variables. The strategic constraint is represented by a constraint function that depends upon a subset of the operational variables. The primary objective function is optimized over a range of target values for the constraint function. An outcome of the primary objective function is determined for each target value, and a graphical view of the outcome for each of the target values is presented. The graphical view provides a visualization of the effect of the strategic constraint on the primary goal of the enterprise.

APPENDIX D

This Appendix contains 5 drawing sheets containing a clean copy of each of the amended Figures 8, 11, 13, 16, and the proposed replacement Figure 12.

APPENDIX E

This Appendix contains a copy of relevant sections of W. Press et al., Numerical Recipes: The Art of Scientific Computing, Cambridge University press, New York (1992), that disclose "simulated annealing", the "Van Wijngaarden-Decker-Brent method", and "Neville's algorithm."



NUMERICAL RECIPES IN C

William H. Press
Brian P. Flannery
Saul A. Teukolsky
William T. Vetterling

CAMBRIDGE

Numerical Recipes in C

The Art of Scientific Computing

William H. Press
Harvard-Smithsonian Center for Astrophysics

Brian P. Flannery
EXXON Research and Engineering Company

Saul A. Teukolsky
Department of Physics, Cornell University

William T. Vetterling
Polaroid Corporation

The right of the University of Cambridge to print and sell all manner of books was granted by Henry VIII in 1534. The University has printed and published continuously since 1584.

CAMBRIDGE UNIVERSITY PRESS
Cambridge
New York Port Chester
Melbourne Sydney

Published by the Press Syndicate of the University of Cambridge
The Pitt Building, Trumpington Street, Cambridge CB2 1RP
40 West 20th Street, New York, NY 10011, USA
10 Stamford Road, Oakleigh, Melbourne 3166, Australia

First published 1988
Reprinted 1988 (twice), 1989, 1990

Printed in the United States of America
Typeset in $\TeX$

The computer programs in this book are available in several machine-readable formats. To purchase diskettes in IBM-compatible personal computer format, use the order form at the back of the book or write to Cambridge University Press, 510 North Avenue, New Rochelle, NY 10801. Also available from Cambridge University Press is the *Numerical Recipes Example Book (C)* which provides demonstration programs that illustrate the use of each program in this book.

This book, the Example Book, and the diskettes are also available in editions in the `FORTRAN` and `Pascal` languages.

Technical questions, corrections, and requests for information on other available formats and additional software products should be directed to Numerical Recipes Software, P.O. Box 243, Cambridge, MA 02238. Inquiries regarding OEM and site licenses should also be directed to this address.

Library of Congress Cataloging-in-Publication Data
Numerical recipes in C.
1. C (Computer program language)
I. Press, William H.
QA76.73.C15N865 1988 005.13'3 87-33844
ISBN 0-521-35465-X book
ISBN 0-521-35466-8 C diskette
ISBN 0-521-35746-2 Example book in C
ISBN 0-521-35467-6 Example diskette

British Library Cataloguing in Publication Data applied for.

Then one can easily derive from (3.1.2) the relations

$$
\begin{aligned}
D_{m+1,i} &= \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \\
C_{m+1,i} &= \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}
\end{aligned}
\qquad (3.1.5)
$$

At each level m, the C's and D's are the corrections which make the interpolation one order higher. The final answer $P_{1\ldots N}$ is equal to the sum of *any* y_i plus a set of C's and/or D's which form a path through the family tree to the rightmost daughter.

Here is a routine for polynomial interpolation or extrapolation from N input points. Note that the input arrays are assumed to be unit-offset. If you have zero-offset arrays, remember to subtract 1 (see §1.2):

```
#include <math.h>

void polint(xa,ya,n,x,y,dy)
float xa[],ya[],x,*y,*dy;
int n;
Given arrays xa[1..n] and ya[1..n], and given a value x, this routine returns a value y, and
an error estimate dy. If P(x) is the polynomial of degree n - 1 such that P(xa_i) = ya_i, i =
1,...,n, then the returned value y = P(x).
{
    int i,m,ns=1;
    float den,dif,dift,ho,hp,w;
    float *c,*d,*vector();
    void nrerror(),free_vector();

    dif=fabs(x-xa[1]);
    c=vector(1,n);
    d=vector(1,n);
    for (i=1;i<=n;i++) {          Here we find the index ns of the closest table entry,
        if ( (dift=fabs(x-xa[i])) < dif) {
            ns=i;
            dif=dift;
        }
        c[i]=ya[i];               and initialize the tableau of c's and d's.
        d[i]=ya[i];
    }
    *y=ya[ns--];                  This is the initial approximation to y.
    for (m=1;m<n;m++) {               For each column of the tableau,
        for (i=1;i<=n-m;i++) {        we loop over the current c's and d's and update them.
            ho=xa[i]-x;
            hp=xa[i+m]-x;
            w=c[i+1]-d[i];
            if ( (den=ho-hp) == 0.0) nrerror("Error in routine POLINT");
            This error can occur only if two input xa's are (to within roundoff) identical.
            den=w/den;
            d[i]=hp*den;          Here the c's and d's are updated.
            c[i]=ho*den;
        }
        *y += (*dy=(2*ns < (n-m) ? c[ns+1] : d[ns--]));
        After each column in the tableau is completed, we decide which correction, c or d, we want to add
        to our accumulating value of y, i.e. which path to take through the tableau—forking up or down.
        We do this in such a way as to take the most "straight line" route through the tableau to its apex,
        updating ns accordingly to keep track of where we are. This route keeps the partial approximations
        centered (insofar as possible) on the target x. The last dy added is thus the error indication.
    }
    free_vector(d,1,n);
    free_vector(c,1,n);
}
```

REFERENCES AND FURTHER READING:

Abramowitz, Milton, and Stegun, Irene A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, vol. 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York), §25.2.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §2.1.

Gear, C. William. 1971, *Numerical Initial Value Problems in Ordinary Differential Equations* (Englewood Cliffs, N.J.: Prentice-Hall), §6.1.

3.2 Rational Function Interpolation and Extrapolation

Some functions are not well approximated by polynomials, but *are* well approximated by rational functions, that is quotients of polynomials. We denote by $R_{i(i+1)\ldots(i+m)}$ a rational function passing through the $m+1$ points $(x_i, y_i) \ldots (x_{i+m}, y_{i+m})$. More explicitly, suppose

$$R_{i(i+1)\ldots(i+m)} = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{p_0 + p_1 x + \cdots + p_\mu x^\mu}{q_0 + q_1 x + \cdots + q_\nu x^\nu} \qquad (3.2.1)$$

Since there are $\mu+\nu+1$ unknown p's and q's (q_0 being arbitrary), we must have

$$m + 1 = \mu + \nu + 1 \qquad (3.2.2)$$

In specifying a rational function interpolating function, you must give the desired order of both the numerator and the denominator.

Rational functions are superior to polynomials, roughly speaking, because of their ability to model functions with poles, that is, zeros of the denominator of equation (3.2.1). These poles might occur for real values of x, if the function to be interpolated itself has poles. More often, the function $f(x)$ is finite for all finite *real* x, but has an analytic continuation with poles in the complex x-plane. Such poles can themselves ruin a polynomial approximation, even one restricted to real values of x, just as they can ruin the convergence of an infinite power series in x. If you draw a circle in the complex plane around your m tabulated points, then you should not expect polynomial interpolation to be good unless the nearest pole is rather far outside the circle. A rational function approximation, by contrast, will stay "good" as long as it has enough powers of x in its denominator to account for (cancel) any nearby poles.

For the interpolation problem, a rational function is constructed so as to go through a chosen set of tabulated functional values. However, we should also mention in passing that rational function approximations can be used in analytic work. One sometimes constructs a rational function approximation by the criterion that the rational function of equation (3.2.1) itself have a power series expansion that agrees with the first $m+1$ terms of the power series expansion of the desired function $f(x)$. This is called *Padé approximation.* It can be quite a powerful technique for turning local information, the derivatives at a point which determine the power series coefficients, into a global capability for evaluating the function, approximately. The approximation is often found to be *remarkably* good.

Bulirsch and Stoer found an algorithm of the Neville type which performs rational function extrapolation on tabulated data. A tableau like that of equation (3.1.2) is constructed column by column, leading to a result and an error estimate. The Bulirsch-Stoer algorithm produces the so-called *diagonal* rational function, with the degrees of numerator and denominator equal (if m is even) or with the degree of the denominator larger by one (if m is odd, cf. equation 3.2.2 above). For the derivation of the algorithm, refer to Stoer and Bulirsch. The algorithm is summarized by a recurrence relation exactly analogous to equation (3.1.3) for polynomial approximation:

$$R_{i(i+1)\dots(i+m)} = R_{(i+1)\dots(i+m)} + \frac{R_{(i+1)\dots(i+m)} - R_{i\dots(i+m-1)}}{\left(\frac{x-x_i}{x-x_{i+m}}\right)\left(1 - \frac{R_{(i+1)\dots(i+m)} - R_{i\dots(i+m-1)}}{R_{(i+1)\dots(i+m)} - R_{(i+1)\dots(i+m-1)}}\right) - 1} \tag{3.2.3}$$

This recurrence generates the rational functions through $m+1$ points from the ones through m and (second term in the denominator of the denominator of equation 3.2.3) $m-1$ points. It is started with

$$R_i = y_i \tag{3.2.4}$$

and with

$$R \equiv [R_{i(i+1)\dots(i+m)} \quad \text{with} \quad m = -1] = 0 \tag{3.2.5}$$

Now, exactly as in equations (3.1.4) and (3.1.5) above, we can convert the recurrence (3.2.3) to one involving only the small differences

$$\begin{aligned} C_{m,i} &\equiv R_{i\dots(i+m)} - R_{i\dots(i+m-1)} \\ D_{m,i} &\equiv R_{i\dots(i+m)} - R_{(i+1)\dots(i+m)} \end{aligned} \tag{3.2.6}$$

Note that these satisfy the relation

$$C_{m+1,i} - D_{m+1,i} = C_{m,i+1} - D_{m,i} \tag{3.2.7}$$

which is useful in proving the recurrences

$$\begin{aligned} D_{m+1,i} &= \frac{C_{m,i+1}(C_{m,i+1} - D_{m,i})}{\left(\frac{x-x_i}{x-x_{i+m+1}}\right) D_{m,i} - C_{m,i+1}} \\ C_{m+1,i} &= \frac{\left(\frac{x-x_i}{x-x_{i+m+1}}\right) D_{m,i}(C_{m,i+1} - D_{m,i})}{\left(\frac{x-x_i}{x-x_{i+m+1}}\right) D_{m,i} - C_{m,i+1}} \end{aligned} \tag{3.2.8}$$

This recurrence is implemented in the following function, whose use is analogous in every way to polint in §3.1. Note again that unit-offset input arrays are assumed (§1.2).

```
#include <math.h>

#define TINY 1.0e-25
#define FREERETURN {free_vector(d,1,n);free_vector(c,1,n);return;}

void ratint(xa,ya,n,x,y,dy)
float xa[],ya[],x,*y,*dy;
int n;
```
Given arrays xa[1..n] and ya[1..n], and given a value x, this routine returns a value of y and an accuracy estimate dy. The value returned is that of the diagonal rational function, evaluated at x, which passes through the n points (xa$_i$,ya$_i$), i = 1...n.
```
{
    int m,i,ns=1;
    float w,t,hh,h,dd,*c,*d,*vector();
    void nrerror(),free_vector();

    c=vector(1,n);
    d=vector(1,n);
    hh=fabs(x-xa[1]);
    for (i=1;i<=n;i++) {
        h=fabs(x-xa[i]);
        if (h == 0.0) {
            *y=ya[i];
            *dy=0.0;
            FREERETURN
        } else if (h < hh) {
            ns=i;
            hh=h;
        }
        c[i]=ya[i];
        d[i]=ya[i]+TINY;        The TINY part is needed to prevent a rare zero-over-zero condi-
    }                               tion.
    *y=ya[ns--];
    for (m=1;m<n;m++) {
        for (i=1;i<=n-m;i++) {
            w=c[i+1]-d[i];
            h=xa[i+m]-x;        h will never be zero, since this was tested in the initializing loop.
            t=(xa[i]-x)*d[i]/h;
            dd=t-c[i+1];
            if (dd == 0.0) nrerror("Error in routine RATINT");    This error condi-
            dd=w/dd;            tion indicates that the interpolating function has a pole at the
            d[i]=c[i+1]*dd;     requested value of x.
            c[i]=t*dd;
        }
        *y += (*dy=(2*ns < (n-m) ? c[ns+1] : d[ns--]));
    }
    FREERETURN
}
```

REFERENCES AND FURTHER READING:

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §2.2.

Gear, C. William. 1971, *Numerical Initial Value Problems in Ordinary Differential Equations* (Englewood Cliffs, N.J.: Prentice-Hall), §6.2.

3.3 Cubic Spline Interpolation

Given a tabulated function $y_i = y(x_i)$, $i = 1...N$, focus attention on one particular interval, between x_j and x_{j+1}. Linear interpolation in that interval gives the interpolation formula

$$y = Ay_j + By_{j+1} \tag{3.3.1}$$

where

$$A \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j} \qquad B \equiv 1 - A = \frac{x - x_j}{x_{j+1} - x_j} \tag{3.3.2}$$

Equations (3.3.1) and (3.3.2) are a special case of the general Lagrange interpolation formula (3.1.1).

Now suppose that, in addition to the tabulated values of y_i, we also have tabulated values for another function denoted y'', that is, a set of numbers

y''_i. We will see in a moment that y'' is supposed to be the second derivative of the function y, but pretend for now that you don't know this.

We next decide, inscrutably for now, to use the values y''_j and y''_{j+1} as linear coefficients of two linearly independent *cubic* polynomial terms which will not spoil the agreement with the tabulated functional values y_j and y_{j+1} at the endpoints x_j and x_{j+1}, *for any choice of y''_j and y''_{j+1}*. A little thought shows that there is only one way to arrange this, namely replacing (3.3.1) by

$$y = Ay_j + By_{j+1} + Cy''_j + Dy''_{j+1} \tag{3.3.3}$$

where A and B are defined in (3.3.2) and

$$C \equiv \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2 \qquad D \equiv \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2 \tag{3.3.4}$$

Notice that the dependence on the independent variable x in equations (3.3.3) and (3.3.4) is entirely through the linear x-dependence of A and B, and (through A and B) the cubic x-dependence of C and D. The reason that (3.3.4) is unique (up to the choice of multiplicative constants in the definition of C and D) is that (i) it is a cubic polynomial in x, (ii) it contains four adjustable linear coefficients, $y_j, y_{j+1}, y''_j, y''_{j+1}$, (iii) four is the correct number of linear coefficients necessary to define a general cubic polynomial, and (iv) four is also the sum of the number of constraints (2, the endpoint values) plus free parameters (2, the numerical values of y''_j and y''_{j+1}).

Now we can see that y'' is in fact the second derivative of the interpolating polynomial. We take derivatives of equation (3.3.3) with respect to x, using the definitions of A, B, C, D to compute $dA/dx, dB/dx, dC/dx$, and dD/dx. The result is

$$\frac{dy}{dx} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{3A^2 - 1}{6}(x_{j+1} - x_j)y''_j + \frac{3B^2 - 1}{6}(x_{j+1} - x_j)y''_{j+1} \tag{3.3.5}$$

for the first derivative, and

$$\frac{d^2y}{dx^2} = Ay''_j + By''_{j+1} \tag{3.3.6}$$

for the second derivative. Since $A = 1$ at x_j, $A = 0$ at x_{j+1}, while B is just the other way around, (3.3.6) shows that y'' is just a tabulated second derivative, and also that the second derivative will be continuous across (e.g.) the boundary between the two intervals (x_{j-1}, x_j) and (x_j, x_{j+1}).

Thus far, we could put in *any* numbers we choose for the y''_i's. However, for a "random" choice of numbers, the value of the *first* derivative, computed from equation (3.3.5), would *not* be continuous across the boundary between two intervals. The key idea of a cubic spline is to require this continuity and to use it to get equations for the numbers y_i''.

The required equations are obtained by setting equation (3.3.5) evaluated for $x = x_j$ in the interval (x_{j-1}, x_j) equal to the same equation evaluated for $x = x_j$ but in the interval (x_j, x_{j+1}). With some rearrangement, this gives (for $j = 2, \ldots N-1$)

$$\frac{x_j - x_{j-1}}{6} y_{j-1}'' + \frac{x_{j+1} - x_{j-1}}{3} y_j'' + \frac{x_{j+1} - x_j}{6} y_{j+1}'' = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}} \tag{3.3.7}$$

These are $N-2$ linear equations in the N unknowns $y_i'', i = 1, \ldots N$. Therefore there is a two parameter family of possible solutions.

For a unique solution, we need to specify two further conditions, typically taken as boundary conditions at x_1 and x_N. The most common ways of doing this are either

- set one or both of y_1'' and y_N'' equal to zero, giving the so-called *natural cubic spline*, which has zero second derivative on one or both of its boundaries, or
- set either of y_1'' and y_N'' to values calculated from equation (3.3.5) so as to make the first derivative of the interpolating function have a specified value on either or both boundaries.

One reason that cubic splines are especially practical is that the set of equations (3.3.7), along with the two additional boundary conditions, are not only linear, but also *tridiagonal*. Each y_j'' is coupled only to its nearest neighbors at $j \pm 1$. Therefore, the equations can be solved in $O(N)$ operations by the tridiagonal algorithm (§2.6). That algorithm is concise enough to build right into the spline calculational routine. This makes the routine not completely transparent as an implementation of (3.3.7), so we encourage you to study it carefully, comparing with `tridag` (§2.6). Arrays are assumed to be unit-offset. If you have zero-offset arrays, see §1.2.

```
void spline(x,y,n,yp1,ypn,y2)
float x[],y[],yp1,ypn,y2[];
int n;
```

Given arrays `x[1..n]` and `y[1..n]` containing a tabulated function, i.e. $y_i = f(x_i)$, with $x_1 < x_2 < \ldots < x_N$, and given values `yp1` and `ypn` for the first derivative of the interpolating function at points 1 and `n`, respectively, this routine returns an array `y2[1..n]` that contains the second derivatives of the interpolating function at the tabulated points x_i. If `yp1` and/or `ypn` are equal to 1×10^{30} or larger, the routine is signalled to set the corresponding boundary condition for a natural spline, with zero second derivative on that boundary.

```
{
    int i,k;
    float p,qn,sig,un,*u,*vector();
    void free_vector();

    u=vector(1,n-1);
    if (yp1 > 0.99e30)                 The lower boundary condition is set either to be "natural"
        y2[1]=u[1]=0.0;
```

```
    else {                             or else to have a specified first derivative.
        y2[1] = -0.5;
        u[1]=(3.0/(x[2]-x[1]))*((y[2]-y[1])/(x[2]-x[1])-yp1);
    }
    for (i=2;i<=n-1;i++) {             This is the decomposition loop of the tridiagonal algorithm.
        sig=(x[i]-x[i-1])/(x[i+1]-x[i-1]);        y2 and u are used for temporary storage
        p=sig*y2[i-1]+2.0;                        of the decomposed factors.
        y2[i]=(sig-1.0)/p;
        u[i]=(y[i+1]-y[i])/(x[i+1]-x[i]) - (y[i]-y[i-1])/(x[i]-x[i-1]);
        u[i]=(6.0*u[i]/(x[i+1]-x[i-1])-sig*u[i-1])/p;
    }
    if (ypn > 0.99e30)                 The upper boundary condition is set either to be "natural"
        qn=un=0.0;
    else {                             or else to have a specified first derivative.
        qn=0.5;
        un=(3.0/(x[n]-x[n-1]))*(ypn-(y[n]-y[n-1])/(x[n]-x[n-1]));
    }
    y2[n]=(un-qn*u[n-1])/(qn*y2[n-1]+1.0);
    for (k=n-1;k>=1;k--)               This is the backsubstitution loop of the tridiagonal algorithm.
        y2[k]=y2[k]*y2[k+1]+u[k];
    free_vector(u,1,n-1);
}
```

It is important to understand that the program `spline` is called only *once* to process an entire tabulated function in arrays x_i and y_i. Once this has been done, values of the interpolated function for any value of x are obtained by calls (as many as desired) to a separate routine `splint` (for "*spl*ine *int*erpolation"):

```
void splint(xa,ya,y2a,n,x,y)
float xa[],ya[],y2a[],x,*y;
int n;
```

Given the arrays `xa[1..n]` and `ya[1..n]`, which tabulate a function (with the xa_i's in order), and given the array `y2a[1..n]`, which is the output from `spline` above, and given a value of `x`, this routine returns a cubic-spline interpolated value `y`.

```
{
    int klo,khi,k;
    float h,b,a;
    void nrerror();

    klo=1;                             We will find the right place in the table by means of bisection.
    khi=n;                                 This is optimal if sequential calls to this routine are at ran-
    while (khi-klo > 1) {                  dom values of x. If sequential calls are in order, and closely
        k=(khi+klo) >> 1;                  spaced, one would do better to store previous values of klo
        if (xa[k] > x) khi=k;              and khi and test if they remain appropriate on the next call.
        else klo=k;
    }                                  klo and khi now bracket the input value of x.
    h=xa[khi]-xa[klo];
    if (h == 0.0) nrerror("Bad XA input to routine SPLINT");    The xa's must be dis-
    a=(xa[khi]-x)/h;                                                tinct.
    b=(x-xa[klo])/h;                   Cubic spline polynomial is now evaluated.
    *y=a*ya[klo]+b*ya[khi]+((a*a*a-a)*y2a[klo]+(b*b*b-b)*y2a[khi])*(h*h)/6.0;
}
```

REFERENCES AND FURTHER READING:

Forsythe, George E., Malcolm, Michael A., and Moler, Cleve B. 1977, *Computer Methods for Mathematical Computations* (Englewood Cliffs, N.J.: Prentice-Hall), §§4.4–4.5.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §2.4.

Ralston, Anthony, and Rabinowitz, Philip. 1978, *A First Course in Numerical Analysis*, 2nd ed. (New York: McGraw-Hill), §3.8.

3.4 How to Search an Ordered Table

Suppose that you have decided to use some particular interpolation scheme, such as fourth-order polynomial interpolation, to compute a function $f(x)$ from a set of tabulated x_i's and f_i's. Then you will need a fast way of finding your place in the table of x_i's, given some particular value x at which the function evaluation is desired. This problem is not properly one of numerical analysis, but it occurs so often in practice that it would be negligent of us to ignore it.

Formally, the problem is this: Given an array of abscissas xx[j], j=1, 2, ..., n, with the elements either monotonically increasing or monotonically decreasing, and given a number x, find an integer j such that x lies between xx[j] and xx[j+1]. For this task, let us define fictitious array elements x[0] and x[n+1] equal to plus or minus infinity (in whichever order is consistent with the monotonicity of the table). Then j will always be between 0 and n, inclusive; a value of 0 indicates "off-scale" at one end of the table, n indicates off-scale at the other end.

In most cases, when all is said and done, it is hard to do better than *bisection*, which will find the right place in the table in about $\log_2 n$ tries. We already did use bisection in the spline evaluation routine splint of the preceding section, so you might glance back at that. Standing by itself, a bisection routine looks like this:

```
void locate(xx,n,x,j)
float xx[],x;
int n,*j;
Given an array xx[1..n], and given a value x, returns a value j such that x is between xx[j]
and xx[j+1]. xx must be monotonic, either increasing or decreasing. j=0 or j=n is returned
to indicate that x is out of range.
{
    int ascnd,ju,jm,jl;

    jl=0;                              Initialize lower
    ju=n+1;                            and upper limits.
    ascnd=xx[n] > xx[1];
    while (ju-jl > 1) {                If we are not yet done,
        jm=(ju+jl) >> 1;               compute a midpoint,
        if (x > xx[jm] == ascnd)
            jl=jm;                     and replace either the lower limit
```

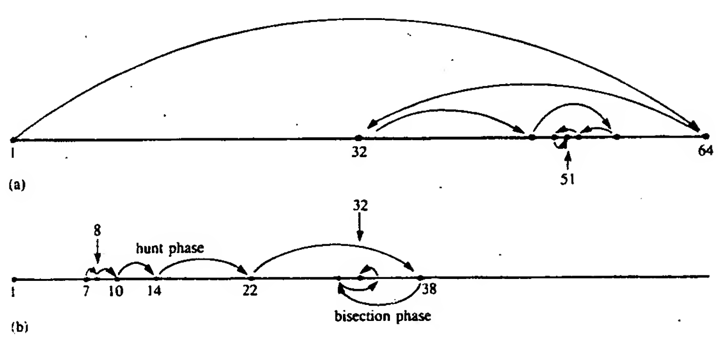


Figure 3.4.1. (a) The routine locate finds a table entry by bisection. Shown here is the sequence of steps that converge to element 51 in a table of length 64. (b) The routine hunt searches from a previous known position in the table by increasing steps, then converges by bisection. Shown here is a particularly unfavorable example, converging to element 32 from element 7. A favorable example would be convergence to an element near 7, such as 9, which would require just three "hops."

```
        else
            ju=jm;                     or the upper limit, as appropriate.
    }                                  Repeat until the test condition is satisfied.
    *j=jl;                             Then set the output
}                                      and return.
```

A unit-offset array xx is assumed. To use locate with a zero-offset array, remember to subtract 1 from the address of xx, and also from the returned value j.

Search with Correlated Values

Sometimes you will be in the situation of searching a table many times, and with nearly identical abscissas on consecutive searches. For example, you may be generating a function that is used on the right-hand side of a differential equation: Most differential-equation integrators, as we shall see in Chapter 15, call for right-hand side evaluations at points that hop back and forth a bit, but whose trend moves slowly in the direction of the integration.

In such cases it is wasteful to do a full bisection, *ab initio*, on each call. The following routine instead starts with a guessed position in the table. It first "hunts", either up or down, in increments of 1, then 2, then 4, etc., until the desired value is bracketed. Second, it then bisects in the bracketed interval. At worst, this routine is about a factor of 2 slower than locate above (if the hunt phase expands to include the whole table). At best, it can be a factor of $\log_2 n$ faster than locate, if the desired point is usually quite close to the input guess. Figure 3.4.1 compares the two routines.

```
        } else {
            del=xh-rtf;
            xh=rtf;
            fh=f;
        }
        dx=xh-xl;                       Convergence.
        if (fabs(del) < xacc || f == 0.0) return rtf;
    }
    nrerror("Maximum number of iterations exceeded in RTFLSP");
}
```

```
#include <math.h>

#define MAXIT 30                    Maximum allowed number of iterations.

float rtsec(func,x1,x2,xacc)
float x1,x2,xacc;
float (*func)();    /* ANSI: float (*func)(float); */
Using the secant method, find the root of a function func thought to lie between x1 and x2.
The root, returned as rtsec, is refined until its accuracy is ±xacc.
{
    int j;
    float fl,f,dx,swap,xl,rts;
    void nrerror();

    fl=(*func)(x1);
    f=(*func)(x2);
    if (fabs(fl) < fabs(f)) {          Pick the bound with the smaller function value as the
        rts=x1;                            most recent guess.
        xl=x2;
        swap=fl;
        fl=f;
        f=swap;
    } else {
        xl=x1;
        rts=x2;
    }
    for (j=1;j<=MAXIT;j++) {           Secant loop.
        dx=(xl-rts)*f/(f-fl);          Increment with respect to latest value.
        xl=rts;
        fl=f;
        rts += dx;
        f=(*func)(rts);                Convergence.
        if (fabs(dx) < xacc || f == 0.0) return rts;
    }
    nrerror("Maximum number of iterations exceeded in RTSEC");
}
```

REFERENCES AND FURTHER READING:

Ralston, Anthony, and Rabinowitz, Philip. 1978, *A First Course in Numerical Analysis*, 2nd ed. (New York: McGraw-Hill), §8.3.

Ostrowski, A.M. 1966, *Solutions of Equations and Systems of Equations*, 2nd ed. (New York: Academic Press), Chapter 12.

9.3 Van Wijngaarden–Dekker–Brent Method

While secant and false position formally converge faster than bisection, one finds in practice pathological functions for which bisection converges more rapidly. These can be choppy, discontinuous functions, or even smooth functions if the second derivative changes sharply near the root. Bisection always halves the interval, while secant and false position can sometimes spend many cycles slowly pulling distant bounds closer to a root. Is there anything we can do to get the best of both worlds?

Yes. We can keep track of whether a supposedly superlinear method is actually converging the way it is supposed to, and, if it is not, we can intersperse bisection steps so as to guarantee *at least* linear convergence. This kind of super-strategy requires attention to bookkeeping detail, and also careful consideration of how roundoff errors can affect the guiding strategy. Also, we must be able to determine reliably when convergence has been achieved.

An excellent algorithm that pays close attention to these matters was developed in the 1960s by van Wijngaarden, Dekker, and others at the Mathematical Center in Amsterdam, and later improved by Brent (reference below). For brevity, we refer to the final form of the algorithm as *Brent's method.* The method is *guaranteed* (by Brent) to converge, so long as the function can be evaluated within the initial interval known to contain a root.

Brent's method combines root bracketing, bisection, and *inverse quadratic interpolation* to converge from the neighborhood of a zero crossing. While the false position and secant methods assume approximately linear behavior between two prior root estimates, inverse quadratic interpolation uses three prior points to fit an inverse quadratic function (x as a quadratic function of y) whose value at $y = 0$ is taken as the next estimate of the root x. Of course one must have contingency plans for what to do if the root falls outside of the brackets. Brent's method takes care of all that. If the three point pairs are $[a, f(a)], [b, f(b)], [c, f(c)]$ then the interpolation formula (cf. equation 3.1.1) is

$$x = \frac{[y-f(a)][y-f(b)]c}{[f(c)-f(a)][f(c)-f(b)]} + \frac{[y-f(b)][y-f(c)]a}{[f(a)-f(b)][f(a)-f(c)]} + \frac{[y-f(c)][y-f(a)]b}{[f(b)-f(c)][f(b)-f(a)]} \qquad (9.3.1)$$

Setting y to zero gives a result for the next root estimate, which can be written as

$$x = b + P/Q \qquad (9.3.2)$$

where, in terms of

$$R \equiv f(b)/f(c), \qquad S \equiv f(b)/f(a), \qquad T \equiv f(a)/f(c) \qquad (9.3.3)$$

we have

$$P = S\,[T(R-T)(c-b) - (1-R)(b-a)] \tag{9.3.4}$$

$$Q = (T-1)(R-1)(S-1) \tag{9.3.5}$$

In practice b is the current best estimate of the root and P/Q ought to be a "small" correction. Quadratic methods work well only when the function behaves smoothly; they run the serious risk of giving very bad estimates of the next root or causing machine failure by an inappropriate division by a very small number ($Q \approx 0$). Brent's method guards against this problem by maintaining brackets on the root and checking where the interpolation would land before carrying out the division. When the correction P/Q would not land within the bounds, or when the bounds are not collapsing rapidly enough, the algorithm takes a bisection step. Thus, Brent's method combines the sureness of bisection with the speed of a higher-order method when appropriate. We recommend it as the method of choice for general one-dimensional root finding where a function's values only (and not its derivative or functional form) are available.

```
#include <math.h>

#define ITMAX 100                  Maximum allowed number of iterations, and machine floating
#define EPS 3.0e-8                 point precision.

float zbrent(func,x1,x2,tol)
float x1,x2,tol;
float (*func)();    /* ANSI: float (*func)(float); */
```

Using Brent's method, find the root of a function `func` known to lie between `x1` and `x2`. The root, returned as `zbrent`, will be refined until its accuracy is `tol`.

```
{
    int iter;
    float a=x1,b=x2,c,d,e,min1,min2;
    float fa=(*func)(a),fb=(*func)(b),fc,p,q,r,s,tol1,xm;
    void nrerror();

    if (fb*fa > 0.0) nrerror("Root must be bracketed in ZBRENT");
    fc=fb;
    for (iter=1;iter<=ITMAX;iter++) {
        if (fb*fc > 0.0) {
            c=a;                            Rename a, b, c and adjust bounding interval d.
            fc=fa;
            e=d=b-a;
        }
        if (fabs(fc) < fabs(fb)) {
            a=b;
            b=c;
            c=a;
            fa=fb;
            fb=fc;
            fc=fa;
        }
        tol1=2.0*EPS*fabs(b)+0.5*tol;       Convergence check.
        xm=0.5*(c-b);
```

```
        if (fabs(xm) <= tol1 || fb == 0.0) return b;
        if (fabs(e) >= tol1 && fabs(fa) > fabs(fb)) {
            s=fb/fa;                        Attempt inverse quadratic interpolation.
            if (a == c) {
                p=2.0*xm*s;
                q=1.0-s;
            } else {
                q=fa/fc;
                r=fb/fc;
                p=s*(2.0*xm*q*(q-r)-(b-a)*(r-1.0));
                q=(q-1.0)*(r-1.0)*(s-1.0);
            }
            if (p > 0.0) q = -q;            Check whether in bounds.
            p=fabs(p);
            min1=3.0*xm*q-fabs(tol1*q);
            min2=fabs(e*q);
            if (2.0*p < (min1 < min2 ? min1 : min2)) {
                e=d;                        Accept interpolation.
                d=p/q;
            } else {
                d=xm;                       Interpolation failed, use bisection.
                e=d;
            }
        } else {                            Bounds decreasing too slowly, use bisection.
            d=xm;
            e=d;
        }
        a=b;                                Move last best guess to a.
        fa=fb;
        if (fabs(d) > tol1)                 Evaluate new trial root.
            b += d;
        else
            b += (xm > 0.0 ? fabs(tol1) : -fabs(tol1));
        fb=(*func)(b);
    }
    nrerror("Maximum number of iterations exceeded in ZBRENT");
}
```

REFERENCES AND FURTHER READING:

Brent, Richard P. 1973, *Algorithms for Minimization without Derivatives* (Englewood Cliffs, N.J.: Prentice-Hall), Chapters 3, 4.

Forsythe, George E., Malcolm, Michael A., and Moler, Cleve B. 1977, *Computer Methods for Mathematical Computations* (Englewood Cliffs, N.J.: Prentice-Hall), §7.2.

we are perhaps tolerant of 5 or 6; but we rarely go higher than that unless there is quite rigorous monitoring of estimated errors.

When your table of values contains many more points than the desirable order of interpolation, you must begin each interpolation with a search for the right "local" place in the table. While not strictly a part of the subject of interpolation, this task is important enough (and often enough botched) that we devote §3.4 to its discussion.

The routines given for interpolation are also routines for extrapolation. An important application, in Chapter 15, is their use in the integration of ordinary differential equations. There, considerable care *is* taken with the monitoring of errors. Otherwise, the dangers of extrapolation cannot be overemphasized: An interpolating function, which is perforce an extrapolating function, will typically go berserk when the argument x is outside the range of tabulated values by more than the typical spacing of tabulated points.

Interpolation can be done in more than one dimension, e.g. for a function $f(x, y, z)$. Multidimensional interpolation is generally accomplished by a sequence of one-dimensional interpolations. We discuss this in §3.6.

REFERENCES AND FURTHER READING:

Abramowitz, Milton, and Stegun, Irene A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, vol. 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York), §25.2.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), Chapter 2.

Acton, Forman S. 1970, *Numerical Methods That Work* (New York: Harper and Row), Chapter 3.

Johnson, Lee W., and Riess, R. Dean. 1982, *Numerical Analysis*, 2nd ed. (Reading, Mass.: Addison-Wesley), Chapter 5.

Ralston, Anthony, and Rabinowitz, Philip. 1978, *A First Course in Numerical Analysis*, 2nd ed. (New York: McGraw-Hill), Chapter 3.

Isaacson, Eugene, and Keller, Herbert B. 1966, *Analysis of Numerical Methods* (New York: Wiley), Chapter 6.

3.1 Polynomial Interpolation and Extrapolation

Through any two points there is a unique line. Through any three points, a unique quadratic. Et cetera. The interpolating polynomial of degree $N - 1$ through the N points $y_1 = f(x_1), y_2 = f(x_2), \ldots y_N = f(x_N)$ is given explicitly by Lagrange's classical formula,

$$P(x) = \frac{(x - x_2)(x - x_3)\ldots(x - x_N)}{(x_1 - x_2)(x_1 - x_3)\ldots(x_1 - x_N)} y_1 + \frac{(x - x_1)(x - x_3)\ldots(x - x_N)}{(x_2 - x_1)(x_2 - x_3)\ldots(x_2 - x_N)} y_2 +$$

There are N terms, each a polynomial of degree $N - 1$ and each construct to be zero for all of the x_i except one, which is constructed to be y_i.

It is not terribly wrong to implement the Lagrange formula straightf wardly, but it is not terribly right either. The resulting algorithm gives error estimate, and it is also somewhat awkward to program. A much bet algorithm (for constructing the same – unique – interpolating polynomial) *Neville's algorithm*, closely related to and sometimes confused with *Aitke algorithm*, the latter now considered obsolete.

Let P_1 be the value at x of the unique polynomial of degree zero (i a constant) passing through the point (x_1, y_1); so $P_1 = y_1$. Likewise defi $P_2, P_3, \ldots P_N$. Now let P_{12} be the value at x of the unique polynomial of deg one passing through both (x_1, y_1) and (x_2, y_2). Likewise $P_{23}, P_{34}, \ldots P_{(N-1)}$ Similarly, for higher order polynomials, up to $P_{123\ldots N}$, which is the value of t unique interpolating polynomial through all N points, i.e. the desired answ The various P's form a "tableau" with "ancestors" on the left leading t single "descendant" at the extreme right. For example, with $N = 4$,

$$\begin{array}{lllll}
x_1: & y_1 = P_1 & & & \\
 & & P_{12} & & \\
x_2: & y_2 = P_2 & & P_{123} & \\
 & & P_{23} & & P_{1234} \\
x_3: & y_3 = P_3 & & P_{234} & \\
 & & P_{34} & & \\
x_4: & y_4 = P_4 & & &
\end{array} \qquad (3.1$$

Neville's algorithm is a recursive way of filling in the numbers in t tableau a column at a time, from left to right. It is based on the relations between a "daughter" P and its two "parents,"

$$P_{i(i+1)\ldots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\ldots(i+m}}{x_i - x_{i+m}} \qquad (3.1$$

This recurrence holds because the two parents already agree at points x_{i+1} x_{i+m-1}.

An improvement on the recurrence (3.1.3) is to keep track of the sm *differences* between parents and daughters, namely to define (for $m = 1, 2$ $N - 1$), we are perhaps tolerant of 5 or 6; but we rarely go higher than that unless there is quite rigorous monitoring of estimated errors.

When your table of values contains many more points than the desirable order of interpolation, you must begin each interpolation with a search for the right "local" place in the table. While not strictly a part of the subject of interpolation, this task is important enough (and often enough botched) that we devote §3.4 to its discussion.

The routines given for interpolation are also routines for extrapolation. An important application, in Chapter 15, is their use in the integration of ordinary differential equations. There, considerable care *is* taken with the monitoring of errors. Otherwise, the dangers of extrapolation cannot be overemphasized: An interpolating function, which is perforce an extrapolating function, will typically go berserk when the argument x is outside the range of tabulated values by more than the typical spacing of tabulated points.

Interpolation can be done in more than one dimension, e.g. for a function $f(x, y, z)$. Multidimensional interpolation is generally accomplished by a sequence of one-dimensional interpolations. We discuss this in §3.6.

REFERENCES AND FURTHER READING:

Abramowitz, Milton, and Stegun, Irene A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, vol. 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York), §25.2.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), Chapter 2.

Acton, Forman S. 1970, *Numerical Methods That Work* (New York: Harper and Row), Chapter 3.

Johnson, Lee W., and Riess, R. Dean. 1982, *Numerical Analysis*, 2nd ed. (Reading, Mass.: Addison-Wesley), Chapter 5.

Ralston, Anthony, and Rabinowitz, Philip. 1978, *A First Course in Numerical Analysis*, 2nd ed. (New York: McGraw-Hill), Chapter 3.

Isaacson, Eugene, and Keller, Herbert B. 1966, *Analysis of Numerical Methods* (New York: Wiley), Chapter 6.

3.1 Polynomial Interpolation and Extrapolation

Through any two points there is a unique line. Through any three points, a unique quadratic. Et cetera. The interpolating polynomial of degree $N-1$ through the N points $y_1 = f(x_1), y_2 = f(x_2), \dots y_N = f(x_N)$ is given explicitly by Lagrange's classical formula,

$$P(x) = \frac{(x-x_2)(x-x_3)\dots(x-x_N)}{(x_1-x_2)(x_1-x_3)\dots(x_1-x_N)}y_1 + \frac{(x-x_1)(x-x_3)\dots(x-x_N)}{(x_2-x_1)(x_2-x_3)\dots(x_2-x_N)}y_2 + \dots + \frac{(x-x_1)(x-x_2)\dots(x-x_{N-1})}{(x_N-x_1)(x_N-x_2)\dots(x_N-x_{N-1})}y_N \tag{3.1.1}$$

There are N terms, each a polynomial of degree $N-1$ and each constructed to be zero for all of the x_i except one, which is constructed to be y_i.

It is not terribly wrong to implement the Lagrange formula straightforwardly, but it is not terribly right either. The resulting algorithm gives no error estimate, and it is also somewhat awkward to program. A much better algorithm (for constructing the same – unique – interpolating polynomial) is *Neville's algorithm*, closely related to and sometimes confused with *Aitken's algorithm*, the latter now considered obsolete.

Let P_1 be the value at x of the unique polynomial of degree zero (i.e. a constant) passing through the point (x_1, y_1); so $P_1 = y_1$. Likewise define $P_2, P_3, \dots P_N$. Now let P_{12} be the value at x of the unique polynomial of degree one passing through both (x_1, y_1) and (x_2, y_2). Likewise $P_{23}, P_{34}, \dots P_{(N-1)N}$. Similarly, for higher order polynomials, up to $P_{123\dots N}$, which is the value of the unique interpolating polynomial through all N points, i.e. the desired answer. The various P's form a "tableau" with "ancestors" on the left leading to a single "descendant" at the extreme right. For example, with $N = 4$,

$$\begin{array}{lllll}
x_1: & y_1 = P_1 & & & \\
 & & P_{12} & & \\
x_2: & y_2 = P_2 & & P_{123} & \\
 & & P_{23} & & P_{1234} \\
x_3: & y_3 = P_3 & & P_{234} & \\
 & & P_{34} & & \\
x_4: & y_4 = P_4 & & &
\end{array} \tag{3.1.2}$$

Neville's algorithm is a recursive way of filling in the numbers in the tableau a column at a time, from left to right. It is based on the relationship between a "daughter" P and its two "parents,"

$$P_{i(i+1)\dots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\dots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\dots(i+m)}}{x_i - x_{i+m}} \tag{3.1.3}$$

This recurrence holds because the two parents already agree at points $x_{i+1} \dots x_{i+m-1}$.

An improvement on the recurrence (3.1.3) is to keep track of the small *differences* between parents and daughters, namely to define (for $m = 1, 2, \dots N-1$),

$$\begin{aligned} C_{m,i} &\equiv P_{i\dots(i+m)} - P_{i\dots(i+m-1)} \\ D_{m,i} &\equiv P_{i\dots(i+m)} - P_{(i+1)\dots(i+m)}. \end{aligned} \tag{3.1.4}$$

We have a degeneracy.

```
ip+1] [kp+1];
+1] [kp+1];
```

urn with message.

variable (see text).

```
kp+1];
```

entioned

al form with N variables and
m with M variables and N
, in essence, the transpose of
nal) problem. It is possible to
primal. This can occasionally
o big deal.
valent to the simplex method
variables are exchanged. Its
that of the simplex method.
equiring only a matrix of size
e stages. If you have a lot of
en you should look into it.
ite simplex algorithm are two
of the usual simplex method:

Progress is made simultaneously towards finding a feasible solution and finding an optimal solution. There seems to be no clearcut evidence that these methods are superior to the usual method by any factor substantially larger than the "tender-loving-care factor" (which reflects the programming effort of the proponents).

Problems where the objective function and/or one or more of the constraints are replaced by expressions nonlinear in the variables are called *nonlinear programming problems.* The literature on such problems is vast, but outside our scope. The special case of quadratic expressions is called *quadratic programming.* Optimization problems where the variables take on only integer values are called *integer programming* problems, a special case of *discrete optimization* generally. The next section looks at a particular kind of discrete optimization problem.

REFERENCES AND FURTHER READING:

Bland, R.G. 1981, *Scientific American*, vol. 244, (June) p. 126.

Kolata, G. 1982, *Science*, vol. 217, p. 39.

Cooper, L., and Steinberg, D. 1970, *Introduction to Methods of Optimization* (Philadelphia: Saunders).

Dantzig, G.B. 1963, *Linear Programming and Extensions* (Princeton, N.J.: Princeton University Press).

Gass, S.T. 1969, *Linear Programming*, 3rd ed. (New York: McGraw-Hill).

Murty, K.G. 1976, *Linear and Combinatorial Programming* (New York: Wiley).

Land, A.H., and Powell, S. 1973, *Fortran Codes for Mathematical Programming* (London: Wiley-Interscience).

Kuenzi, H.P., Tzschach, H.G., and Zehnder, C.A. 1971 *Numerical Methods of Mathematical Optimization* (New York: Academic Press).

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §4.10.

Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer-Verlag).

10.9 Combinatorial Minimization: Method of Simulated Annealing

The *method of simulated annealing* is a technique that has recently attracted significant attention as suitable for optimization problems of very large scale. For practical purposes, it has effectively "solved" the famous *traveling salesman problem* of finding the shortest cyclical itinerary for a traveling salesman who must visit each of N cities in turn. The method has also been used successfully for designing complex integrated circuits: The arrangement of several hundred thousand circuit elements on a tiny silicon substrate is optimized so as to minimize interference among their connecting wires. Amazingly, the implementation of the algorithm is quite simple.

Notice that the two applications cited are both examples of *combinatorial minimization.* There is an objective function to be minimized, as usual; but the space over which that function is defined is not simply the N-dimensional space of N continuously variable parameters. Rather, it is a discrete, but very large, configuration space, like the set of possible orders of cities, or the set of possible allocations of silicon "real estate" to circuit elements. The number of elements in the configuration space is factorially large, so that they cannot be explored exhaustively. Furthermore, since the set is discrete, we are deprived of any notion of "continuing downhill in a favorable direction." The concept of "direction" may not have any meaning in the configuration space.

At the heart of the method of simulated annealing is an analogy with thermodynamics, specifically with the way that liquids freeze and crystallize, or metals cool and anneal. At high temperatures, the molecules of a liquid move freely with respect to one another. If the liquid is cooled slowly, thermal mobility is lost. The atoms are often able to line themselves up and form a pure crystal that is completely ordered over a distance up to billions of times the size of an individual atom in all directions. This crystal is the state of minimum energy for this system. The amazing fact is that, for slowly cooled systems, nature is able to find this minimum energy state. In fact, if a liquid metal is cooled quickly or "quenched," it does not reach this state but rather ends up in a polycrystalline or amorphous state having somewhat higher energy.

So the essence of the process is *slow* cooling, allowing ample time for redistribution of the atoms as they lose mobility. This is the technical definition of *annealing*, and it is essential for ensuring that a low energy state will be achieved.

Although the analogy is not perfect, there is a sense in which all of the minimization algorithms thus far in this chapter correspond to rapid cooling or quenching. In all cases, we have gone greedily for the quick, nearby solution: from the starting point, go immediately downhill as far as you can go. This, as often remarked above, leads to a local, but not necessarily a global, minimum. Nature's own minimization algorithm is based on quite a different procedure. The so-called Boltzmann probability distribution,

$$\text{Prob}\,(E) \sim \exp(-E/kT) \qquad (10.9.1)$$

expresses the idea that a system in thermal equilibrium at temperature T has its energy probabilistically distributed among all different energy states E. Even at low temperature, there is a chance, albeit very small, of a system being in a high energy state. Therefore, there is a corresponding chance for the system to get out of a local energy minimum in favor of finding a better, more global, one. The quantity k (Boltzmann's constant) is a constant of nature which relates temperature to energy. In other words, the system sometimes goes *uphill* as well as downhill; but the lower the temperature, the less likely is any significant uphill excursion.

In 1953, Metropolis and coworkers first incorporated these kinds of principles into numerical calculations. Offered a succession of options, a simulated

thermodynamic system was assumed to change its configuration from energy E_1 to energy E_2 with probability $p = \exp[-(E_2 - E_1)/kT]$. Notice that if $E_2 < E_1$, this probability is greater than unity; in such cases the change is arbitrarily assigned a probability $p = 1$, i.e. the system *always* took such an option. This general scheme, of always taking a downhill step while *sometimes* taking an uphill step, has come to be known as the Metropolis algorithm.

To make use of the Metropolis algorithm for other than thermodynamic systems, one must provide the following elements:

1. A description of possible system configurations.
2. A generator of random changes in the configuration; these changes are the "options" presented to the system.
3. An objective function E (analog of energy) whose minimization is the goal of the procedure.
4. A control parameter T (analog of temperature) and an *annealing schedule* which tells how it is lowered from high to low values, e.g., after how many random changes in configuration is each downward step in T taken, and how large is that step. The meaning of "high" and "low" in this context, and the assignment of a schedule, may require physical insight and/or trial-and-error experiments.

The Traveling Salesman Problem

A concrete illustration is provided by the traveling salesman problem. The salesperson visits N cities with given positions (x_i, y_i), returning finally to his or her city of origin. Each city is to be visited only once, and the route is to be made as short as possible. This problem belongs to a class known as *NP-complete* problems, whose computation time for an *exact* solution increases with N as exp(const. $\times N$), becoming rapidly prohibitive in cost as N increases. The traveling salesman problem also belongs to a class of minimization problems for which the objective function E has many local minima. In practical cases, it is often enough to be able to choose from these a minimum which, even if not absolute, cannot be significantly improved upon. The annealing method manages to achieve this, while limiting its calculations to scale as a small power of N.

As a problem in simulated annealing, the traveling salesman problem is handled as follows:

1. *Configuration.* The cities are numbered $i = 1 \ldots N$ and each has coordinates (x_i, y_i). A configuration is a permutation of the number $1 \ldots N$, interpreted as the order in which the cities are visited.

2. *Rearrangements.* An efficient set of moves has been suggested by Lin. The moves consist of two types: (a) A section of path is removed and then replaced with the same cities running in the opposite order; or (b) a section of path is removed and then replaced in between two cities on another, randomly chosen, part of the path.

3. *Objective Function.* In the simplest form of the problem, E is taken just as the total length of journey,

$$E = L \equiv \sum_{i=1}^{N} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \qquad (10.9.2)$$

with the convention that point $N+1$ is identified with point 1. To illustrate the flexibility of the method, however, we can add the following additional wrinkle: suppose that the salesman has an irrational fear of flying over the Mississippi River. In that case, we would assign each city a parameter μ_i, equal to +1 if it is east of the Mississippi, −1 if it is west, and take the objective function to be

$$E = \sum_{i=1}^{N} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} + \lambda(\mu_i - \mu_{i+1})^2 \qquad (10.9.3)$$

A penalty 4λ is thereby assigned to any river crossing. The algorithm now finds the shortest path that avoids crossings. The relative importance that it assigns to length of path versus river crossings is determined by our choice of λ. Figure 10.9.1 shows the results obtained. Clearly, this technique can be generalized to include many conflicting goals in the minimization.

4. *Annealing schedule.* This requires experimentation. We first generate some random rearrangements, and use them to determine the range of values of ΔE that will be encountered from move to move. Choosing a starting value for the parameter T which is considerably larger than the largest ΔE normally enountered, we proceed downward in multiplicative steps each amounting to a 10 percent decrease in T. We hold each new value of T constant for, say, $100N$ reconfigurations, or for $10N$ successful reconfigurations, whichever comes first. When efforts to reduce E further become sufficiently discouraging, we stop.

The following traveling salesman program, using the Metropolis algorithm, should illustrate for you the important aspects of the simulated annealing technique.

```
#include <stdio.h>
#include <math.h>

#define TFACTR 0.9               Annealing schedule – t is reduced by this factor on each step.
#define ALEN(a,b,c,d) sqrt(((b)-(a))*((b)-(a))+((d)-(c))*((d)-(c)))

void anneal(x,y,iorder,ncity)
float x[],y[];
int iorder[],ncity;
```
This algorithm finds the shortest round-trip path to `ncity` cities whose coordinates are in the arrays `x[1..ncity]`, `y[1..ncity]`. The array `iorder[1..ncity]` specifies the order in which the cities are visited. On input, the elements of `iorder` may be set to any permutation of the numbers `1` to `ncity`. This routine will return the best alternative path it can find.
```
{
    int ans,nover,nlimit,i1,i2,idum;
    unsigned long int iseed;
```

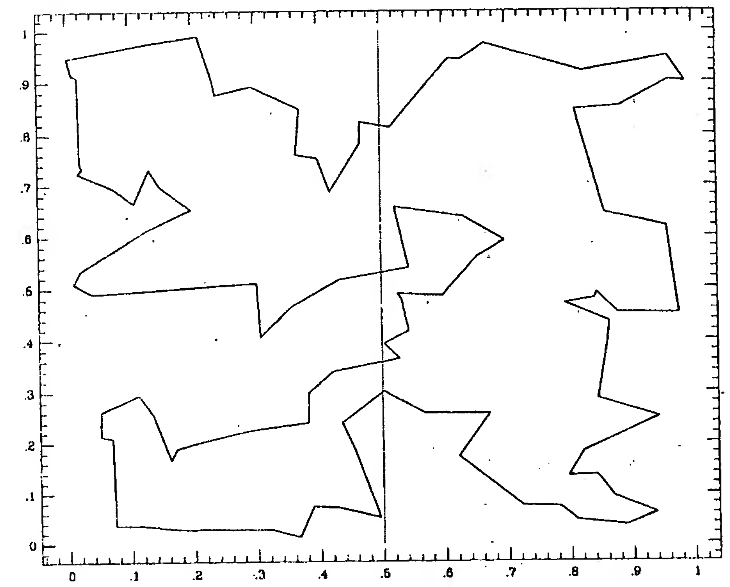

(a)

Figure 10.9.1. Traveling salesman problem solved by simulated annealing. The (nearly) shortest path among 100 randomly positioned cities is shown in (a). The dotted line is a river, but there is no penalty in crossing. In (b) the river-crossing penalty is made large, and the solution restricts itself to the minimum number of crossings, two. In (c) the penalty has been made negative: the salesman is actually a smuggler who crosses the river on the flimsiest excuse!

```
    int i,j,k,nsucc,nn,idec;
    static int n[7];
    float path,de,t;
    float ran3();
    int irbit1(),metrop();
    void reverse(),trnspt();
    float revcst(),trncst();

    nover=100*ncity;                 Maximum number of paths tried at any temperature.
    nlimit=10*ncity;                 Maximum number of successful path changes before
    path=0.0;                            continuing.
    t=0.5;
    for (i=1;i<ncity;i++) {          Calculate initial path length.
        i1=iorder[i];
        i2=iorder[i+1];
        path += ALEN(x[i1],x[i2],y[i1],y[i2]);
    }
    i1=iorder[ncity];                Close the loop by tying path ends together.
    i2=iorder[1];
    path += ALEN(x[i1],x[i2],y[i1],y[i2]);
    idum = -1;
    iseed=111;
    for (j=1;j<=100;j++) {           Try up to 100 temperature steps.
        nsucc=0;
        for (k=1;k<=nover;k++) {
```

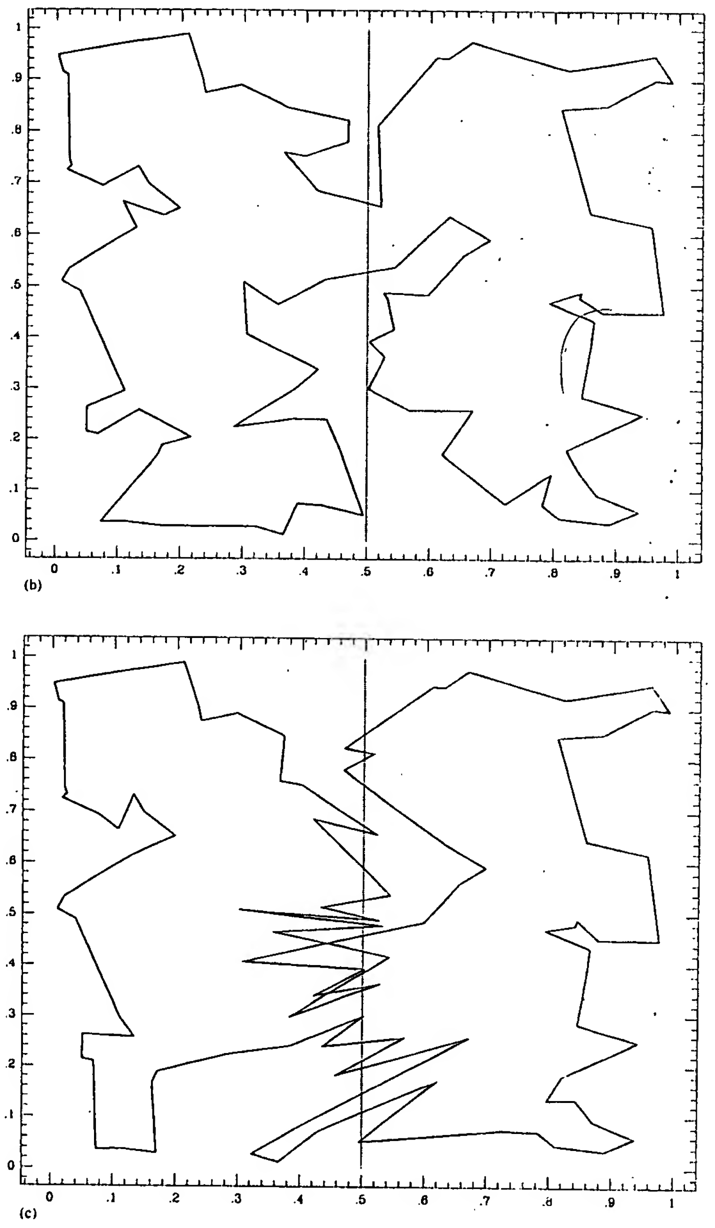


Figure 10.9.1. Traveling salesman problem solved by simulated annealing (see caption on previous page).

```
	do {
		n[1]=1+(int) (ncity*ran3(&idum));          Choose beginning of segment ..
		n[2]=1+(int) ((ncity-1)*ran3(&idum));          ..and end of segment.
		if (n[2] >= n[1]) ++n[2];
		nn=1+((n[1]-n[2]+ncity-1) % ncity);          nn is the number of cities not on
	} while (nn<3);                                        the segment.
	idec=irbit1(&iseed);                 Decide whether to do a segment reversal or transport.
	if (idec == 0) {                     Do a transport.
		n[3]=n[2]+(int) (abs(nn-2)*ran3(&idum))+1;
		n[3]=1+((n[3]-1) % ncity);             Transport to a location not on the path.
		de=trncst(x,y,iorder,ncity,n);         Calculate cost.
		ans=metrop(de,t);                      Consult the oracle.
		if (ans) {
			++nsucc;
			path += de;
			trnspt(iorder,ncity,n);        Carry out the transport.
		}
	} else {                             Do a path reversal
		de=revcst(x,y,iorder,ncity,n);         Calculate cost.
		ans=metrop(de,t);                      Consult the oracle.
		if (ans) {
			++nsucc;
			path += de;
			reverse(iorder,ncity,n);       Carry out the reversal.
		}
	}
	if (nsucc >= nlimit) break;       Finish early if we have enough successful changes.
	}
	printf("\n %s %10.6f %s %12.6f \n","T =",t,
		"   Path Length =",path);
	printf("Successful Moves: %6d\n",nsucc);
	t *= TFACTR;                         Annealing schedule.
	if (nsucc == 0) return;              If no success, we are done.
	}
}

float revcst(x,y,iorder,ncity,n)
float x[],y[];
int iorder[],ncity,n[];
```

This function returns the value of the cost function for a proposed path reversal. ncity is the number of cities, and arrays x[1..ncity], y[1..ncity] give the coordinates of these cities. iorder[1..ncity] holds the present itinerary. The first two values n[1] and n[2] of array n give the starting and ending cities along the path segment which is to be reversed. On output, de is the cost of making the reversal. The actual reversal is not performed by this routine.

```
{
	float xx[5],yy[5],de;
	int j,ii;

	n[3]=1 + ((n[1]+ncity-2) % ncity);          Find the city before n[1] ..
	n[4]=1 + (n[2] % ncity);                    .. and the city after n[2]
	for (j=1;j<=4;j++) {
		ii=iorder[n[j]];                    Find coordinates for the four cities involved.
		xx[j]=x[ii];
		yy[j]=y[ii];
	}
	de = -ALEN(xx[1],xx[3],yy[1],yy[3]);        Calculate cost of disconnecting the seg-
	de -= ALEN(xx[2],xx[4],yy[2],yy[4]);            ment at both ends and reconnecting
	de += ALEN(xx[1],xx[4],yy[1],yy[4]);            in the opposite order.
	de += ALEN(xx[2],xx[3],yy[2],yy[3]);
	return de;
}

void reverse(iorder,ncity,n)
int iorder[],ncity,n[];
```

This routine performs a path segment reversal. **iorder[1..ncity]** is an input array giving the present itinerary. The vector **n** has as its first four elements the first and last cities **n[1]**,**n[2]** of the path segment to be reversed, and the two cities **n[3]** and **n[4]** which immediately precede and follow this segment. **n[3]** and **n[4]** are found by function **revcst**. On output, **iorder** contains the segment from **n[1]** to **n[2]** in reversed order.

```
{
    int nn,j,k,l,itmp;

    nn=(1+((n[2]-n[1]+ncity) % ncity))/2;      This many cities must be swapped to ef-
                                                   fect the reversal.
    for (j=1;j<=nn;j++) {                       Start at the ends of the segment and
        k=1 + ((n[1]+j-2) % ncity);                swap pairs of cities, moving toward
        l=1 + ((n[2]-j+ncity) % ncity);            the center.
        itmp=iorder[k];
        iorder[k]=iorder[l];
        iorder[l]=itmp;
    }
}
```

```
float trncst(x,y,iorder,ncity,n)
float x[],y[];
int iorder[],ncity,n[];
```

This function returns the value of the cost function for a proposed path segment transport. **ncity** is the number of cities, and arrays **x[1..ncity]** and **y[1..ncity]** give the city coordinates. **iorder[1..ncity]** is an array giving the present itinerary. The first three elements of array **n** give the starting and ending cities of the path to be transported, and the point among the remaining cities after which it is to be inserted. On output, **de** is the cost of the change. The actual transport is not performed by this routine.

```
{
    float xx[7],yy[7],de;
    int j,ii;

    n[4]=1 + (n[3] % ncity);                    Find the city following n[3]..
    n[5]=1 + ((n[1]+ncity-2) % ncity);          ..and the one preceding n[1]..
    n[6]=1 + (n[2] % ncity);                    ..and the one following n[2].
    for (j=1;j<=6;j++) {
        ii=iorder[n[j]];                        Determine coordinates for the six cities in-
        xx[j]=x[ii];                               volved.
        yy[j]=y[ii];
    }
    de = -ALEN(xx[2],xx[6],yy[2],yy[6]);        Calculate the cost of disconnecting the
    de -= ALEN(xx[1],xx[5],yy[1],yy[5]);           path segment from n[1] to n[2], open-
    de -= ALEN(xx[3],xx[4],yy[3],yy[4]);           ing a space between n[3] and n[4],
    de += ALEN(xx[1],xx[3],yy[1],yy[3]);           connecting the segment in the space,
    de += ALEN(xx[2],xx[4],yy[2],yy[4]);           and connecting n[5] to n[6].
    de += ALEN(xx[5],xx[6],yy[5],yy[6]);
    return de;
}
```

```
void trnspt(iorder,ncity,n)
int iorder[],ncity,n[];
```

This routine does the actual path transport, once **metrop** has approved. **iorder[1..ncity]** is an input array giving the present itinerary. The array **n** has as its six elements the beginning **n[1]** and end **n[2]** of the path to be transported, the adjacent cities **n[3]** and **n[4]** between which the path is to be placed, and the cities **n[5]** and **n[6]** which precede and follow the path. **n[4]**, **n[5]** and **n[6]** are calculated by function **trncst**. On output, **iorder** is modified to reflect the movement of the path segment.

```
{
    int m1,m2,m3,nn,j,jj,*jorder,*ivector();
    void free_ivector();

    jorder=ivector(1,ncity);
    m1=1 + ((n[2]-n[1]+ncity) % ncity);         Find the number of cities from n[1] to n[2] ..
    m2=1 + ((n[5]-n[4]+ncity) % ncity);         ..and the number from n[4] to n[5] ..
    m3=1 + ((n[3]-n[6]+ncity) % ncity);         ..and the number from n[6] to n[3].
```

```
    nn=1;
    for (j=1;j<=m1;j++) {
        jj=1 + ((j+n[1]-2) % ncity);            Copy the chosen segment.
        jorder[nn++]=iorder[jj];
    }
    if (m2>0) {
        for (j=1;j<=m2;j++) {                   Then copy the segment from n[4] to n[5].
            jj=1+((j+n[4]-2) % ncity);
            jorder[nn++]=iorder[jj];
        }
    }
    if (m3>0) {
        for (j=1;j<=m3;j++) {                   Finally, the segment from n[6] to n[3].
            jj=1 + ((j+n[6]-2) % ncity);
            jorder[nn++]=iorder[jj];
        }
    }
    for (j=1;j<=ncity;j++)
        iorder[j]=jorder[j];                    Copy jorder back into iorder.
    free_ivector(jorder,1,ncity);
}
```

```
int metrop(de,t)
float de,t;
```

Metropolis algorithm. **metrop** returns a boolean variable which issues a verdict on whether to accept a reconfiguration which leads to a change **de** in the objective function **e**. If **de**<0, **metrop** = **TRUE** (1), while if **de**>0, **metrop** is only **TRUE** with probability **exp(-de/t)**, where **t** is a temperature determined by the annealing schedule.

```
{
    static int gljdum=1;
    float ran3();

    return de < 0.0 || ran3(&gljdum) < exp(-de/t);
}
```

Assessing the Promise of Simulated Annealing

There is not yet enough practical experience with the method of simulated annealing to say definitively that it will realize its current promise. The method has several extremely attractive features, rather unique when compared with other optimization techniques.

First, it is not "greedy", in the sense that it is not easily fooled by the quick payoff achieved by falling into unfavorable local minima. Provided that sufficiently general reconfigurations are given, it wanders freely among local minima of depth less than about T. As T is lowered, the number of such minima qualifying for frequent visits is gradually reduced.

Second, configuration decisions tend to proceed in a logical order. Changes which cause the greatest energy differences are sifted over when the control parameter T is large. These decisions become more permanent as T is lowered, and attention then shifts more to smaller refinements in the solution. For example, in the traveling salesman problem with the Mississippi River twist, if λ is large, a decision to cross the Mississippi only twice is made at high T, while the specific routes on each side of the river are determined only at later stages.

The analogies to thermodynamics may be pursued to a greater extent than we have done here. Quantities analogous to specific heat and entropy may be defined, and these can be useful in monitoring the progress of the algorithm toward an acceptable solution. Information on this subject is found in the references by Kirkpatrick et al.

REFERENCES AND FURTHER READING:

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. 1983, *Science*, vol. 220, pp. 671–680.

Kirkpatrick, S. 1984, *Journal of Statistical Physics*, vol. 34, p. 975.

Vecchi, M.P. and Kirkpatrick, S. 1983, *IEEE Transactions on Computer Aided Design*, vol. CAD-2, p. 215.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller A., and Teller, E. 1953 *J. Chem. Phys.*, vol. 21, p. 1087.

Lin, S. 1965, *Bell Syst. Tech. Journ.*, vol. 44, p. 2245.

Christofides, N., Mingozzi, A., Toth, P., and Sandi, C., eds. 1979, *Combinatorial Optimization* (London and New York: Wiley-Interscience) [not simulated annealing, but other topics and algorithms].

Chapter 11. Eigensystems

11.0 Introduction

An $N \times N$ matrix $\mathbf{A}$ is said to have an *eigenvector* $\mathbf{x}$ and corresponding *eigenvalue* λ if

$$\mathbf{A} \cdot \mathbf{x} = \lambda \mathbf{x} \tag{11.0.1}$$

Obviously any multiple of an eigenvector $\mathbf{x}$ will also be an eigenvector, but we won't consider such multiples as being distinct eigenvectors. (The zero vector is not considered to be an eigenvector at all.) Evidently (11.0.1) can hold only if

$$\det |\mathbf{A} - \lambda \mathbf{1}| = 0 \tag{11.0.2}$$

which, if expanded out, is an N^{th} degree polynomial in λ whose roots are the eigenvalues. This proves that there are always N (not necessarily distinct) eigenvalues. Equal eigenvalues coming from multiple roots are called *degenerate*. Root-searching in the characteristic equation (11.0.2) is usually a very poor computational method for finding eigenvalues. We will learn much better ways in this chapter, as well as efficient ways for finding corresponding eigenvectors.

The above two equations also prove that every one of the N eigenvalues has a (not necessarily distinct) corresponding eigenvector: If λ is set to an eigenvalue, then the matrix $\mathbf{A} - \lambda \mathbf{1}$ is singular, and we know that every singular matrix has at least one nonzero vector in its nullspace (see §2.9 on singular value decomposition).

If you add $\tau \mathbf{x}$ to both sides of (11.0.1), you will easily see that the eigenvalues of any matrix can be changed or *shifted* by an additive constant τ by adding to the matrix that constant times the identity matrix. The eigenvectors are unchanged by this shift. Shifting, as we will see, is an important part of many algorithms for computing eigenvalues. We see also that there is no special significance to a zero eigenvalue. Any eigenvalue can be shifted to zero, or any zero eigenvalue can be shifted away from zero.